Contribution des nouvelles approches de modélisation à la durabilité des applications

Benjamin Chevallereau

École Centrale de Nantes

ÉCOLE DOCTORALE

Science Pour l'Ingénieur, Géosciences, Architecture

Année 2011 N° B.U. : 498-174

Thèse de DOCTORAT

Diplôme délivré par l'Ecole Centrale de Nantes

Membre du PRES LUNAM

Spécialité : GENIE MECANIQUE, GENIE INDUSTRIEL

Présentée et soutenue publiquement par :

BENJAMIN CHEVALLEREAU

Le 11 février 2011

À l'École Centrale de Nantes

TITRE

CONTRIBUTION DES NOUVELLES APPROCHES DE MODELISATION A LA DURABILITE DES APPLICATIONS

JURY

Président :	Pierre Alain MULLER	Professeur des Universités à l'Université de Haute Alsace
Rapporteurs :	Colette ROLLAND	Professeur des Universités à Paris 1 Panthéon Sorbonne
	Lionel ROUCOULES	Professeur des Universités à Arts et Métiers ParisTech
Examinateurs :	Alain BERNARD	Professeur des Universités à l'École Centrale de Nantes
	Pierre MEVELLEC	Professeur des Universités à l'Université de Nantes
	Raphaël CHENOUARD	Maître de conférences à l'École Centrale de Nantes
Invités :	Jean-Christophe KERMAGORET	Directeur associé de la société BlueXML
	Régis BAUDU	Directeur R&D de la société BlueXML

Directeur de thèse : Alain BERNARD / Laboratoire : IRCCyN

Co-directeur de thèse : Pierre MEVELLEC / Laboratoire : LEMNA

Co-Encadrant de thèse : Raphaël CHENOUARD / Laboratoire : IRCCyN

N° ED 498-174

Résumé

Les organisations actuelles se structurent et agissent en s'appuyant sur leurs systèmes d'information. Malgré les progrès considérables réalisés par la technologie informatique, on constate que les acteurs restent très souvent critiques par rapport à leur systèmes d'information. Une des causes de cet écart entre les espoirs et la réalité trouve sa source dans la difficulté à produire un cahier des charges suffisamment détaillé pour les opérationnels et interprétable par les spécialistes des systèmes d'information. Notre proposition vise à surmonter cet obstacle en organisant l'expression des besoins dans un langage commun aux opérationnels et aux experts techniques. Pour cela, le langage proposé pour exprimer les besoins est basé sur la notion de but.

L'ensemble de cette démarche repose sur l'ingénierie dirigée par les modèles. Celle-ci a aujourd'hui montré la majorité de ces résultats dans la phase de développement logiciel et tout particulièrement avec l'approche MDA. Tandis que cette phase est grandement étudiée par la communauté IDM, la phase de spécification et d'expression du besoin est, aujourd'hui, peu approfondie. Notre proposition repose sur la mise en œuvre de l'ingénierie dirigée par les modèles dans cette phase, qui semble être l'une des plus importantes dans le processus de développement logiciel, avec pour objectif d'améliorer la qualité de la spécification des besoins et ainsi apporter une information plus fiable et plus claire aux étapes suivantes. Cette proposition repose sur un méta-modèle de spécification du besoin fonctionnel et d'un mécanisme d'interprétation à l'aide de transformations de modèles.

Title : Contribution of new modelling approaches to software sustainability.

Actual organization are structured and act with the help of their information systems. In spite of considerable progresses made by computer technology, we note that actors are very often critical on their information systems. Difficulties to product specifications enough detailed for functional profile and interpretable by information system expert is one of reason of this gap between hopes and reality. Our proposition wants to get over this obstacle by organizing user requirements in a common language of operational profile and technical expert.

This proposition is based on model driven engineering. It shows, today, the most part of its results in the phase of software development and more particularly with the MDA approach. Whereas this phase is greatly studied by the MDE community, the phase of specification and expression of needs is, today, not many detailed by this community. Our proposition is based on the implementation of MDE to improve the quality of specification of needs and to improve the communication between technical experts and functionnal profiles. This proposition is based on a meta-model, a modeler and a set of interpretations.

Mots-clés: Ingénierie des besoins ; ingénierie des modèles ; spécification fonctionnelle ; méthodologie de développement.

Keywords: Requirements engineering ; model driven engineering ; functional spécification ; development methodology.

« Tu peux tout accomplir dans la vie si tu as le courage de le rêver, l'intelligence d'en faire un projet réaliste, et la volonté de voir ce projet mené à bien. »

Sidney A Friedman

Remerciements

Je voudrais tout d'abord remercier mes directeurs et encadrants de thèse, en commençant par Alain BERNARD pour m'avoir fait confiance pour ce projet et m'encourager à chaque instant. Je remercie également Pierre MEVELLEC pour nos discussions m'apportant un regard différent sur mes travaux de recherche. Je terminerais par Raphaël CHENOUARD qui a su apporté un regard critique et novateur au sujet des mes propositions.

Nous avons pu organiser et réagir rapidement à la soutenance grâce au dynamisme et à l'intérêt que m'ont apporté les différents membres du jury :

- Pierre-Alain MULLER, Professeur à l'Université de Haute Alsace, pour avoir accepté de présider ce jury de thèse ;
- Colette ROLLAND, Professeur à Paris I/Panthéon/Sorbonne, pour les remarques et questions apportées en tant que rapportrice de ce travail ;
- Lionel ROUCOULES, Professeur à Arts et Métier ParisTech, pour ses commentaires pertinents et analogies à la conception de produit.

Cette thèse n'aurait pas pu être réalisée sans le soutien de la société BlueXML. Je tiens donc à remercier vivement les directeurs de cette société : Jean-Christophe KERMAGORET et Éric LE GAREC. Ils ont su me proposer cette thèse, m'aider à l'organiser, me soutenir à chaque instant et m'apporter un regard critique sur mes propositions. Bien évidemment, mes remerciements vont également directement à Régis BAUDU, directeur R&D de la société BlueXML, pour nos discussions et son investissement dans ce projet.

Je n'oublie pas tous ceux qui ont eu un rôle moins officiel mais tout aussi important dans mon apprentissage. Je remercie également vivement les personnes qui m'ont soutenu pendant ces trois années de thèse. Je commence par les membres de l'IRCCyN et plus particulièrement les membres de l'équipe IVGI : Philippe, Florent, Julien, Catherine, Joanna, Roland, François et plus largement Jonathan, Yoann, Benjamin, Raphaël pour tous les bons moments

passés ensemble. Je pense également aux membres passés et actuels de la société BlueXML : Brice, David, Pierre-Charles, Christophe, Éric, Pierre, Estelle…

Je finirais mes remerciements par l'ensemble des personnes qui m'ont soutenu au quotidien et qui m'ont donné confiance pour réaliser ce projet. Je pense à ma famille, mes amis et mes proches…

Tables des matières

INTRODUCTION GENERALE 3

I. DEMARCHE DE CONCEPTION ET DE REPRESENTATION DES BESOINS UTILISATEUR 9

A. Gestion des besoins dans les cycles de conception logicielle 10
1. Modèle en cascade 11
2. Modèle en V 12
3. Modèle en spirale 14
4. Panorama des méthodes agiles et pseudo-agiles 15
5. Synthèse 18

B. Approche de conception centrée utilisateur 19
1. La norme ISO 13407 19
2. Étapes du processus de conception centrée utilisateur 21
   a) Planification du processus de conception 21
   b) Spécification du contexte d'utilisation 22
   c) Spécification des besoins utilisateur 23
   d) Production des solutions 23
   e) Evaluation des solutions 23
   f) Synthèse 24

C. Techniques de recueil et d'analyse des besoins 24
1. Les méthodes d'analyse de l'usage 25
2. Les méthodes d'inspection de l'utilisabilité 28
3. Les méthodes expérimentales 28
4. Les méthodes participatives et créatives 29
5. Synthèse 30

D. Méthodologies en ingénierie des besoins 31
1. KAOS 34
2. i* 35
3. Tropos 37
4. NFR 38
5. Synthèse 39

E. Approche MDA 40
1. Généralités 40
2. Description 40
3. Généralisation à l'ingénierie dirigée par les modèles 43
4. Panorama des solutions de transformation de modèles 45
5. Comparaison d'un processus de développement traditionnel et d'un processus de développement appliquant la méthode MDA 47
6. Synthèse 48

F. Conclusion 49

II. SUN : SUSTAINABLE USER NEED 52

A. Introduction 53

B. Les composantes de la méthodologie SUN 54
1. Le langage 54
a) Présentation 54
b) Syntaxe abstraite 56
c) Syntaxe concrète 60
2. Le mécanisme d'interprétation 65

C. Positionnement de la méthodologie SUN dans le processus de conception 68

D. Les acteurs de la méthodologie SUN 70

E. Étapes de SUN 72
1. Structuration des besoins utilisateur 72
2. Définition du dictionnaire 73
3. Analyse de la structure et de la stratégie de l'organisation 74
4. Spécification des politiques d'accès aux données 74
5. Validation\Annotation par les utilisateurs 75

F. Synthèse 76

III. INTERPRETATIONS DISPONIBLES DANS SUN 80

A. Identification des anomalies 81
1. Méta-modèle 82
2. Transformation de modèles 82
3. Génération 83

B. Approche par les cartes heuristiques ou « mind maps » 85
1. Méta-modèle 86
2. Transformation de modèles 87
3. Génération 90

C. Évaluation des risques 91
1. Méta-modèle 94
2. Transformation de modèles 95
3. Génération 95

D. Prototypage 97

E. Documentation 101
1. Méta-modèle 102
2. Transformation de modèles 103
3. Génération 104

F. Synthèse 106

IV. APPLICATION A LA CONCEPTION D'UN OUTIL DE GESTION DE CONFERENCE 108

A. Modèle de besoin 110

B. Analyse des risques 117

C. Prototypage 118

D. Génération des modèles de conception 119
1. Modèle de donnée 119

2. Modèle de formulaire 121

E. Génération et raffinage sur une solution de gestion documentaire 121

F. Synthèse 122

V. CONCLUSIONS ET PERSPECTIVES 126

VI. BIBLIOGRAPHIE 132

A. Valorisation des travaux de thèse 133

B. Bibliographie externe 134

ANNEXES 142

ANNEXE A MODELES DEFINIS DANS LE TUTORIEL KAOS POUR UN SYSTEME D'ELEVATEUR. 143

ANNEXE B EXEMPLES DE MODELES UTILISANT LE LANGAGE I* 145

ANNEXE C MODELE SAINT AUGUSTIN 147

ANNEXE D MODELE WAGRAM 148

ANNEXE E TRANSFORMATION DE MODELES 149

Table des figures

Figure I-1 : Modèle de conception en cascade 12
Figure I-2 : Modèle de conception en V 13
Figure I-3 : Modèle de conception en spirale 15
Figure I-4 : Le cycle RAD 17
Figure I-5 : Processus de conception centrée utilisateur 21
Figure I-6 : Schéma descriptif de la méthode par entretien 25
Figure I-7 : Schéma descriptif de la méthode par questionnaire 26
Figure I-8 : Schéma descriptif de la méthode par observation 27
Figure I-9 : Schéma descriptif de la méthode par analyse des traces 28
Figure I-10 : Exemple de scénarios utilisés [DAM 06] 33
Figure I-11 : Méta-modèle KAOS 35
Figure I-12 : Méta-modèle i* 36
Figure I-13 : Méta-modèle NFR 39
Figure I-14 : Méta-modèle de l'approche MDA [MIR 06] 41
Figure I-15 : Pyramide de modélisation de l'OMG 45
Figure II-1 : Concepts principaux de méta-modélisation (EMOF 2.0) 58
Figure II-2 : Méta-modèle SUN au format eCore 60
Figure II-3 : Outil de modélisation web 61
Figure II-4 : Modèle de configuration de la syntaxe concrète 62
Figure II-5 : Outil de modélisation basé sur Eclipse 63
Figure II-6 : Récapitulatif des éléments graphiques disponibles 64
Figure II-7 : Fenêtre d'édition des politiques d'accès 65
Figure II-8 : Mécanisme d'interprétation 67
Figure II-9 : Positionnement de SUN dans un cycle en cascade 69
Figure II-10 : Positionnement de SUN dans un cycle en V 69
Figure II-11 : Schéma de communication de la méthode SUN 71
Figure II-12 : Déroulement de la méthode SUN 78
Figure III-1 : Méta-modèle du diagnostic 82
Figure III-2 : Schéma de la transformation vers un diagnostique 83
Figure III-3 : Diagramme d'objets du modèle de diagnostique 83
Figure III-4 : Diagnostique interprété sous la forme d'un tableau 85
Figure III-5 : Méta-modèle de cartes heuristiques 86

Figure III-6 : Schéma de la transformation ATL Req2GoalList 88
Figure III-7 : Diagramme d'objets du modèle de carte conceptuelle issue de la transformation Req2GoalList 89
Figure III-8 : Schéma de la transformation ATL Req2GoalListByAgent 90
Figure III-9 : Résultat de la génération ouvert avec l'outil FreeMind 91
Figure III-10 : Méta-modèle de calcul de risque 94
Figure III-11: Schéma de la transformation ATL Req2Risk 95
Figure III-12 : Estimation des risques obtenue après génération 97
Figure III-13 : Méta-modèle de projet Web 99
Figure III-14 : Illustration de la transformation pour générer un prototype 100
Figure III-15 : Prototype Web généré 101
Figure III-16 : Méta-modèle de documentation 102
Figure III-17: Schéma de la transformation ATL Req2Documentation 103
Figure III-18 : Modèle de documentation 104
Figure IV-1 : Modèle de besoin correspondant à l'outil de gestion de conférence 112
Figure IV-2 : Definition des agents et de leurs responsabilités 113
Figure IV-3 : Carte heuristique représentant la liste des objectifs d'un auteur. 114
Figure IV-4 : Dictionnaire de données du système de gestion de conférence 114
Figure IV-5 : Politiques d'accès définies pour les objectifs sous la responsabilité d'un auteur 116
Figure IV-6 : Carte heuristique synthétisant la vue du dictionnaire de données par un auteur 116
Figure IV-7 : Analyse des risques sur les agents et les entités 117
Figure IV-8 : Analyse des risques sur les objectifs 118
Figure IV-9 : Prototype web obtenu à partir de la spécification du besoin 119
Figure IV-10 : Modèle de données obtenu après transformation 120
Figure IV-11 : Modèle de données obtenu après raffinement 121
Figure IV-12 : Outil développé de gestion de conférences 122
Figure A-1 : Exemple de modèle des buts KAOS 143
Figure A-2 : Exemple de modèle des responsabilités KAOS 143
Figure A-3 : Exemple de modèle objet KAOS 144
Figure A-4 : Exemple de modèles d'opération KAOS 144
Figure B-5 : Exemple de modèle SD proposé par la méthodologie i* [CAS 01] .. 145
Figure B-6 : Exemple de modèle SR proposé par la méthodologie i* [CAS 01] .. 146

Table des tableaux

Tableau I-1 : Synthèse des techniques de recueil des besoins 31
Tableau I-2 : Comparaison du processus de développement traditionnel et approche MDA 47
Tableau IV-1 : Répartition proportionnelle de la charge entre les étapes 124
Tableau IV-2 : Répartition des charges effectives 124

Listings

Listing 1 : Template Acceleo de génération à base de diagnostique 84
Listing 2 : Fichier JSON généré à partir d'un modèle de diagnostique 85
Listing 3 : Template de génération à base de carte heuristique 91
Listing 4 : Template de génération à base d'une estimation des risques 96
Listing 5 : Template de génération à base d'un modèle de documentation 106
Listing 6 : Transformation ATL Req2Diagnostic 149
Listing 7 : Transformation ATL Req2GoalList 155
Listing 8 : Transformation ATL Req2GoalListByAgent 156
Listing 9 : Transformation ATL Req2Risk 158
Listing 10 : Transformation ATL Req2Prototype 161
Listing 11 : Transformation ATL Req2Documentation 167

Glossaire

AADL :	Architecture Analysis & Design Language
AG :	Agile Modeling
AS :	Abstract Syntax
AMMA :	ATLAS Model Management Architecture
API :	Application Programming Interface
ATL :	Atlas Transformation Language
CCU :	Conception Centrée Utilisateur
CIM :	Computation Independent Model
CRUD :	Create, Read, Update, Delete
CS :	Concrete Syntax
DSDM :	Dynamic Systems Development Method
DSL :	Domain Specific Language
E-A :	Entité-Association
EMF :	Eclipse Modeling Framework
EMOF :	Essential Meta Object Facility
GEMS :	Generic Eclipse Modeling Framework
GME :	Global Modeling Environment
GMF :	Graphical Modeling Framework
IB :	Ingénierie des Besoins
IDM :	Ingénierie Dirigée par les Modèles
ISO :	International Organization for Standardization
JSON :	JavaScript Object Notation
KAOS :	Knowledge Acquisition in autOmated Specification Keep All Objects Satisfied
KM3 :	Kernel Meta Meta Model
KTR :	Kermeta Transformation Rule
MDA :	Model Driven Architecture
MDE :	Model Driven Engineering
MOF :	Meta-Object Facility
OCL :	Object Constraint Language
OMG :	Object Management Group
OMT :	Object Modeling Technique
OOSE :	Object Oriented Software Engineering
PDM :	Platform Description Model
PIM :	Platform Independent Model

PSM :	Platform Specific Model
QVT :	Query/View/Transformation
RAD :	Rapid Application Development
SI :	Système d'Information
SII :	Système d'Information Interactif
RUP :	Rational Unified Process
SAM :	Structured Automata Metamodel
SCRAM :	SCenario Requirements Analysis Method
SYSML :	SYStems Modeling Language
TCS :	Textual Concrete Syntax
TIGER :	TransformatIon-based Generation of modeling EnviRonment
TOPCASED :	Toolkit in Open Source for Critical Applications & Systems Development
UML :	Unified Modeling Language
UP :	Unified Process
XML :	Extensible Markup Language
XP :	eXtreme Programming

Introduction générale

Un système d'information (SI) est un ensemble structuré de ressources interconnectées permettant l'acquisition, la gestion et le partage de données[1]. Les informations ainsi manipulées représentent un ensemble de connaissances que le système doit mettre en relation afin de produire un résultat. Celui-ci doit être manipulable et compréhensible par l'ensemble des acteurs du système d'information qu'ils soient de nature humaine ou non. Selon la complexité et la finalité du SI, le problème à résoudre est plus ou moins simple à modéliser.

Un système d'information interactif (SII) est un système d'information pour lequel une personne (ou un groupe de personnes) est à la fois une ressource de donnée et un destinataire de l'information produite [MOU 07]. La finalité d'un SII est de fournir, à chaque profil d'utilisateur, un moyen simple de consommer l'information nécessaire à la réalisation de leurs tâches et d'offrir la possibilité de capturer l'ensemble des informations pertinentes produites.

La performance d'un SII illustre l'adéquation entre les besoins réels des utilisateurs et les moyens fournis par le SI pour consommer et produire de l'information. La norme ISO 9241 [ISO 98] donne les lignes directrices concernant l'utilisabilité. D'après cette norme, « un système est utilisable lorsqu'il permet à l'utilisateur de réaliser sa tâche avec efficacité, efficience et satisfaction dans le contexte d'utilisation spécifié ». Plusieurs chercheurs s'intéressent à l'utilisabilité universelle ([HOC 01], [SCH 00] et [RIE 00]). La notion derrière l'utilisabilité universelle est qu'il n'existe pas d'utilisateur moyen : l'objectif à atteindre vise le plus grand nombre possible d'utilisateurs. Nielsen [NIE 92] propose un modèle pour la conception d'interfaces. Il s'agit du cycle d'utilisabilité (« Usability Engineering Life Cycle »), qui est appliqué de façon itérative. Même si certaines étapes semblent triviales, elles sont trop souvent négligées et c'est l'une des raisons pour lesquelles plusieurs systèmes d'information n'atteignent pas les objectifs fixés au départ [VAU 03]. Dans un

[1] La définition de ce qu'est un système d'information (SI) ne sera pas développée dans cette thèse car cela ne présente aucun apport au travail présenté. Nous nous bornerons donc à la résumer de cette manière et laisserons le lecteur se reporter aux références bibliographiques suivantes : [GAB 04] et [MOR 06].

cycle d'utilisabilité, nous tentons plutôt de connaître les besoins de l'utilisateur en observant et en évaluant les tâches et les processus. Selon Nielsen [NIE 92], la raison de cette pratique est que les utilisateurs ne peuvent pas le plus souvent exprimer correctement leurs besoins.

Le contexte économique actuel oblige les organisations à être créatives et à se surpasser pour faire face à une concurrence de plus en plus féroce. Malgré les nombreux efforts faits en ce sens et la mobilisation des ressources humaines, cela ne suffit plus. L'organisation doit non seulement s'informatiser, mais elle doit faire le bon choix quant à la mise en place d'une infrastructure informatique qui saura répondre à ses besoins présents et futurs. L'utilisabilité d'un SII doit donc être une préoccupation primordiale pendant sa conception. Les organisations actuelles se structurent et agissent en s'appuyant sur leur SI. Malgré les progrès considérables réalisés par la technologie informatique, nous constatons que les acteurs restent très souvent critiques par rapport à leur SI. Une des causes de cet écart entre les espoirs et la réalité trouve sa source dans la difficulté à produire un cahier des charges suffisamment détaillé pour les opérationnels et interprétable par les spécialistes des SI.

À la vue des différentes analyses synthétisées par D. Krob [KRO 05], il est possible d'extraire deux conclusions sur la performance de l'industrie logicielle. Tout d'abord, il est très difficile d'obtenir des mesures fiables de cette performance [GLA 06]. La seconde conclusion nous montre que l'industrie logicielle produit des outils dont la qualité peut être améliorée. Après avoir identifié le problème à résoudre, c'est-à-dire la qualité des produits, d'autres analyses nous permettent d'identifier les principales raisons de ce taux d'échec. La définition des besoins et leur compréhension représentent les problèmes majeurs pour réaliser des produits répondant aux exigences des utilisateurs.

Traditionnellement, l'ingénierie des systèmes d'information se concentre sur la modélisation conceptuelle. Celle-ci vise à abstraire la spécification du système requis à partir de l'analyse des informations nécessaires à la communauté des utilisateurs. Cette spécification se concentre sur ce que doit faire le système, c'est-à-dire ses fonctionnalités [ROL 07]. Bien que la modélisation conceptuelle

permette aux profils techniques de comprendre la sémantique du domaine, elle ne permet pas de construire des systèmes acceptés par la communauté des utilisateurs. En effet, de nombreuses études [COO 01] [EUR 96] [ROB 02] [STA 94] montrent que les systèmes réalisés répondent de manière inadéquate ou insuffisante aux besoins exprimés par les utilisateurs. Les processus de développement actuels comportent donc des faiblesses. Celles-ci impliquent d'adapter les systèmes en cours ou après la réalisation, ce qui oblige à fournir une quantité d'effort très importante [JOH 95]. À ce propos, D.A. Norman, dans son ouvrage [NOR 99], souligne qu'une étude approximative des usages peut conduire à une conception fortement biaisée.

Une des pistes pour expliquer ce fossé est le manque de bases communes de communication entre les disciplines en interaction dans un projet de conception de SI. En effet, les différences de langages d'expression et de modèles de représentation entre les intervenants ne favorisent pas toujours le partage d'information. Certaines méthodes de conception tendent à compenser ce manque par la mise en place de formalismes de description accessibles à tous les acteurs. Cet effort contribue à faciliter le dialogue mais cela reste insuffisant pour l'assimilation des connaissances métier pendant la totalité de la conception du SII.

Cette thèse cherche à répondre aux problèmes de communication entre les différents participants d'un projet de conception de SI. Pour cela, nous proposons une méthodologie outillée de définition des besoins utilisateur. Celle-ci est supportée par l'ingénierie dirigée par les modèles (IDM). L'hypothèse avancée est que le fossé sémantique que nous avons identifié précédemment pourrait être en partie comblé par l'utilisation d'un langage simplifié de spécification du besoin utilisateur accessible par l'ensemble des acteurs de la conception d'un SII mais également d'un mécanisme d'interprétation générique permettant ensuite d'évaluer, analyser, vérifier ou traduire cette spécification.

Le premier chapitre sera consacré à l'analyse des méthodes existantes en conception de SI. Celle-ci portera sur l'identification des méthodes traditionnelles de conception, puis l'approche de conception centrée utilisateur

pour aboutir sur les techniques de recueil des besoins. Cette analyse sera poursuivie en répertoriant les méthodes existantes en ingénierie des besoins (IB) pour terminer sur une présentation de l'approche MDA (Model Driven Architecture).

Le second chapitre propose l'ensemble des composants disponibles dans notre approche. Il y sera défini le langage proposé, l'outil de modélisation ainsi que le mécanisme générique d'interprétation. L'ensemble des étapes de la méthodologie proposée sera également abordé dans ce chapitre.

Le troisième chapitre présente les différentes interprétations réalisées sur la base du mécanisme générique proposé dans le chapitre précédent. Il en ressortira les objectifs, les avantages et les inconvénients de chacune d'elles.

Une application concrète de conception d'un SII sera l'objet du quatrième chapitre. Il y sera défini un outil de gestion de conférences en utilisant l'approche proposée. Il en ressortira une analyse des avantages et inconvénients de la méthode comparée aux approches plus traditionnelles. Afin de montrer la généricité de notre approche, d'autres cas d'études réalisés seront abordés dans différents domaines tels que les systèmes hospitaliers, le domaine de la finance ou encore la dématérialisation de documents.

Enfin une conclusion retracera les grandes lignes de ce manuscrit et soulignera les contributions de notre approche. Diverses perspectives, aussi bien scientifiques qu'industrielles seront alors proposées.

Démarche de conception et de représentation des besoins utilisateur

Ce chapitre présente les différentes approches existantes dans la démarche de conception de SII. Nous commencerons par analyser les cycles traditionnels de conception, que sont le modèle en cascade, le modèle en V ainsi que le modèle en spirale, en nous attardant sur la phase de spécification des besoins utilisateurs. Nous nous intéresserons ensuite aux méthodes agiles qui impliquent, par définition, au maximum le client pour acquérir une grande flexibilité et réactivité.

La partie suivante abordera l'approche de conception centrée utilisateur normalisée par l'ISO dans la norme 13407 [ISO 99]. Celle-ci définit un guide de bonnes pratiques visant à améliorer la prise en compte des utilisateurs pour la conception de systèmes interactifs. Nous y détaillerons les étapes de la norme ainsi que les moments, méthodes et outils préconisés pour leur mise en place.

Ensuite, nous définirons les techniques de recueil des besoins couramment utilisés dans l'industrie pour continuer sur l'étude des différentes méthodes proposées par l'ingénierie des besoins. Celles-ci seront ensuite analysées pour en déduire une grille d'évaluation.

Nous terminerons ce chapitre par la présentation de l'approche MDA proposée par l'OMG. Elle procure une méthode pour l'utilisation de modèles dans la conception, la construction, le déploiement, la maintenance et les modifications de systèmes. Elle permet, via des outils, de spécifier un système indépendamment de la plateforme sur laquelle il sera implanté er transformer les spécifications d'un système pour une plateforme particulière.

A. Gestion des besoins dans les cycles de conception logicielle

P. Jaulent [JAU 94] définit le génie logiciel comme un ensemble de procédures, de méthodes, de langages et d'ateliers imposés ou préconisés par des normes adaptées à l'environnement d'utilisation afin de favoriser la production et la maintenance de composants logiciels de qualité. Le génie logiciel traite du cycle de vie des logiciels à travers toutes ses phases : l'analyse du besoin, l'élaboration des spécifications, l'élaboration de l'architecture, l'implémentation, la phase de

test et la maintenance. Le processus de conception est structuré en différentes phases suivant un modèle de conception.

Nous présenterons dans cette partie les trois modèles principaux de conception : modèle en cascade, modèle en V et modèle en spirale. Nous nous intéresserons ensuite aux méthodes agiles dont l'objectif principal est une forte implication du client pour finir sur la méthode de processus unifié guidée par les besoins des utilisateurs.

1. Modèle en cascade

Le modèle en cascade (cf. Figure 0-1) est le modèle le plus ancien pour la gestion du cycle de vie du logiciel. Celui-ci a été mis au point dés 1966 puis formalisé en 1970. Il a longtemps servi de référence pour la conception de produit [BOE 81]. Le principe de base de ce modèle est de diviser le projet en plusieurs phases distinctes sans retour possible. Lorsqu'une phase est achevée, son résultat sert de point d'entrée à la phase suivante. Suite à l'utilisation de ce modèle, des modifications ont été apportées dans le but de le rendre plus flexible comme la possibilité de retour à l'étape précédente.

Le processus débute par des itérations successives entre l'étape de faisabilité et d'analyse des besoins. A l'issue de cette analyse, le cahier des charges est rédigé et servira de référence pour l'étape suivante de conception. A partir de cette étape, l'utilisateur n'est plus du tout impliqué jusqu'à l'étape d'exploitation. Suivent les étapes de spécification et de modélisation de l'application pour atteindre l'étape de codage. Une fois les tests unitaires validés, le logiciel passe en phase d'intégration et de validation. Après le succès de cette étape, celui-ci peut être installé chez le client et enfin exploité par l'utilisateur final.

Le but de ce modèle est de réduire les risques en proposant une démarche progressive. Le problème majeur de ce modèle est la méthode de validation à chaque fin d'étape en isolant l'utilisateur final dès la phase de conception. Le contrôle du logiciel se fera exclusivement en fin de projet. S'il s'avère que le système ne répond pas aux besoins de l'utilisateur final, il est alors trop tard et

les efforts de correction seront considérables pour proposer une solution acceptable.

Pendant les phases d'étude de faisabilité et d'analyse des besoins, aucune préconisation n'est faite par la méthode pour la modélisation des besoins. Ces étapes reposent majoritairement sur les connaissances des concepteurs qui se doivent de prévoir les différents contextes de fonctionnement du système [KOL 93].

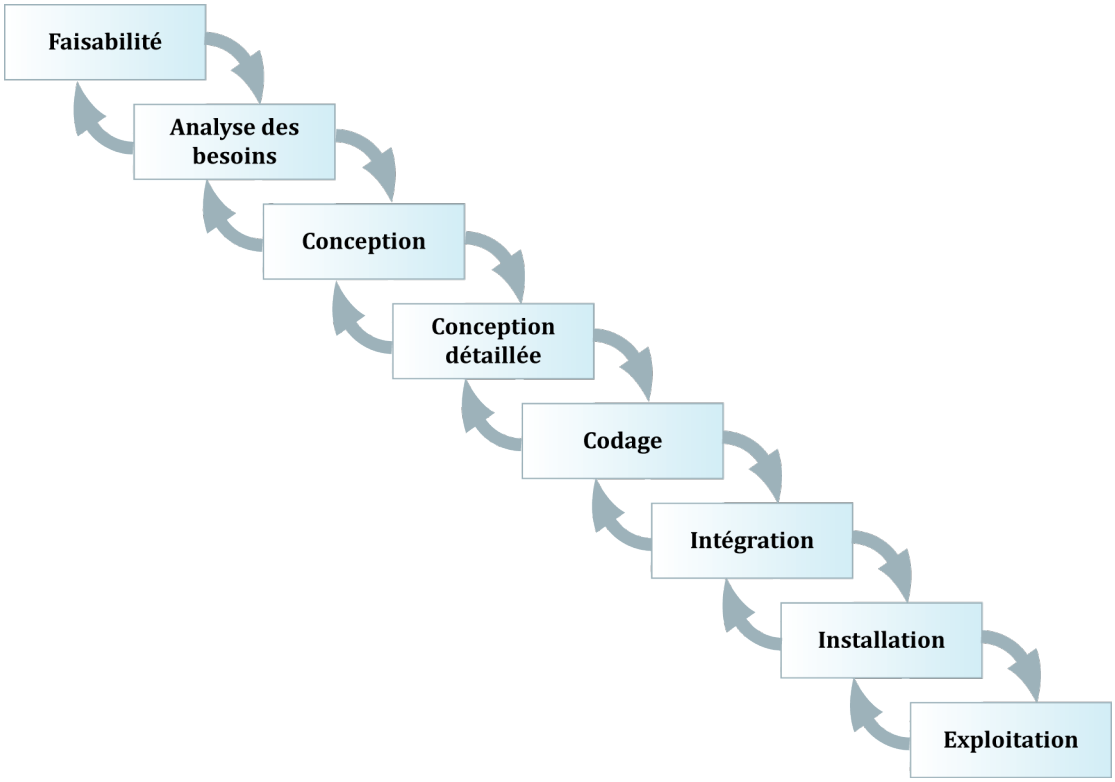


Figure 0-1 : Modèle de conception en cascade

2. Modèle en V

Le modèle en V (Figure 0-2) permet de mettre en avant la détection d'erreur le plus en amont possible lors des étapes de conception. Ce modèle reprend les étapes du modèle en cascade en y ajoutant une confrontation entre les différentes étapes de même niveau. La cohérence entre les deux éléments permet de vérifier en continu que le projet progresse vers un produit répondant aux besoins initiaux.

Nous retrouverons dans le modèle en V les étapes présentées dans le modèle en cascade. Cependant, cette approche est composée de deux pentes. La première est dédiée aux étapes de spécification, de conception et d'implémentation, et la seconde aux étapes de vérification du code obtenu [CAL 90] [BRE 01].

Du point de vue de l'implication de l'utilisateur, comme pour le modèle en cascade, aucune recommandation précise n'est spécifiée quant aux modalités de recueil des besoins. De plus, il n'est pas clairement indiqué quelles étapes font appel aux utilisateurs finaux. Le modèle en V exprime un processus de conception pour lequel chaque étape est réalisée à partir de la précédente et donc introduit un processus dans lequel la validité d'une étape dépend de la validité de l'étape précédente, tout comme le modèle en cascade.

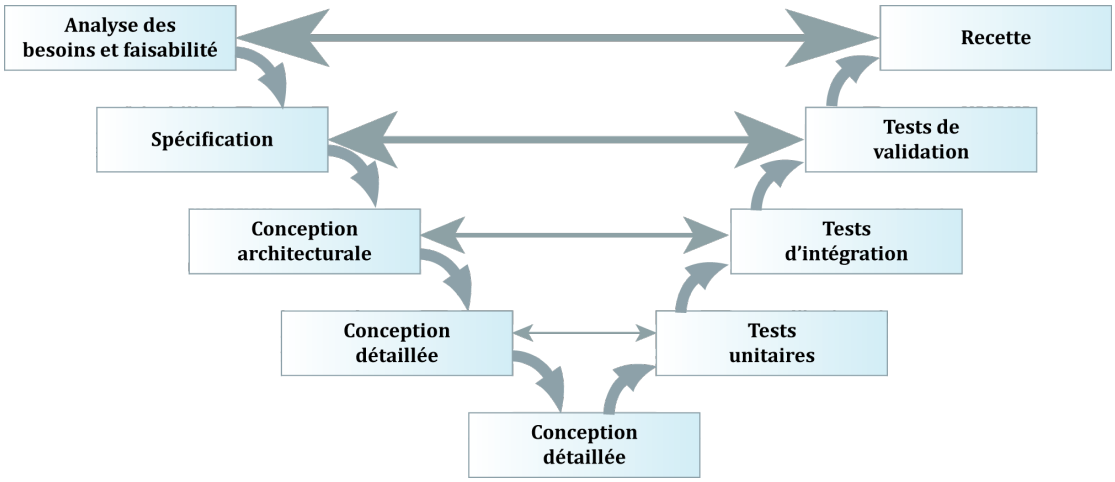


Figure 0-2 : Modèle de conception en V

3. Modèle en spirale

Proposé par D. Boehm [BOE 84], le cycle de vie en spirale est un modèle générique de cycle de vie évolutif. Ce modèle tient compte de la possibilité de réévaluer les risques en cours de développement. Celui-ci se base sur le prototypage incrémental en le complétant d'une prise de décision managériale et non purement technique.

Le modèle en spirale est défini en quatre étapes par D. Boehm :

a. détermination des objectifs du cycle, des alternatives pour les atteindre et des contraintes à partir des résultats des cycles précédents, ou de l'analyse préliminaire des besoins;
b. analyse des risques, évaluation des alternatives à partir de maquettage et/ou prototypage;
c. développement et vérification de la solution retenue, un modèle « classique » (cascade ou en V) peut être utilisé ici ;
d. revue des résultats et vérification du cycle suivant.

Le modèle en spirale favorise l'expression progressive des besoins tout au long des étapes de conception. De plus, le prototypage successif permet une validation des critères ergonomiques tout en réduisant les risques de conception [NIE 93]. Le modèle en spirale s'applique essentiellement en interne, lorsque les clients et les fournisseurs font partie de la même organisation. Dans le cas inverse, c'est-à-dire dans une relation client/fournisseur ordinaire, il y a eu signature de contrat et donc l'effort doit être estimé à l'avance. Le modèle en spirale ne peut donc s'appliquer.

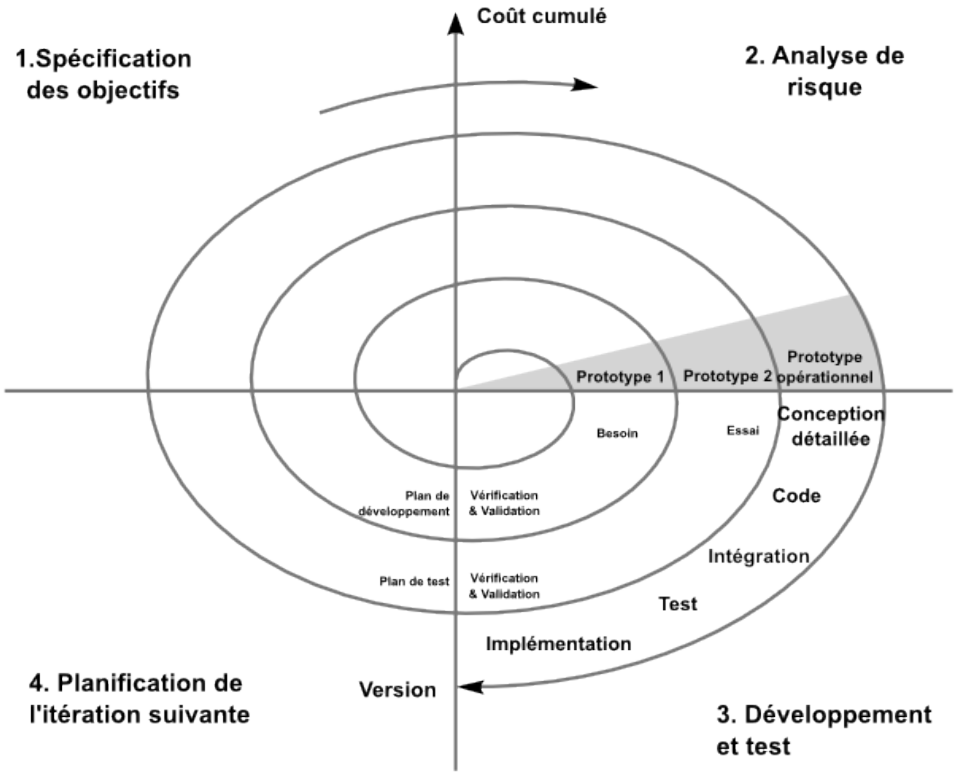


Figure 0-3 : Modèle de conception en spirale

4. Panorama des méthodes agiles et pseudo-agiles

Les méthodes agiles (en anglais « Agile Modeling », noté AG) visent à réduire le cycle de vie du logiciel en développant une version minimale, puis en intégrant les fonctionnalités par un processus itératif basé sur une écoute client et des tests tout au long du cycle de développement, à l'image du modèle en spirale.

La méthode « eXtreme Programming » (XP) [BEC 04] définit un ensemble de bonnes pratiques permettant de développer un logiciel en plaçant l'utilisateur au centre de la démarche de conception. Cette méthode est notamment basée sur des cycles de travail très courts, de l'ordre d'une à deux semaines maximum. De plus, les différentes versions du logiciel sont livrées très tôt et à une fréquence élevée pour optimiser l'impact des remarques des utilisateurs. Ce type de méthode peut permettre d'obtenir des résultats très intéressants mais dans des contextes favorables tels qu'une équipe de développement restreinte.

La méthode « Dynamic Systems Development Method » (DSDM) [STA 97] est née du constat que les temps de développements étaient jusqu'à présents trop longs,

que les logiciels livrés ne correspondaient pas toujours aux besoins des utilisateurs et que les développeurs étaient peu impliqués dans la conception. Cette méthode se base sur neuf principes :

a. L'implication active des utilisateurs est impérative.
b. Les équipes DSDM ont un pouvoir de décision.
c. Livraison fréquente de produits.
d. Livraison des éléments fonctionnels en temps voulu.
e. Un développement itératif et incrémental.
f. Toute modification pendant le développement est réversible.
g. Les besoins sont définis à un niveau global.
h. Les tests sont intégrés à toutes les étapes du cycle de vie.
i. Un esprit de coopération et de collaboration entre tous les acteurs.

Une autre méthode proche des méthodes agiles, mais préconisant une planification rigide, est basée sur un processus de développement nommé « Rapid Application Development » (RAD). Cette méthode, introduite par J. Martin [MAR 91], tend à raccourcir le cycle de vie en remplaçant l'étape de spécification/conception par une phase de prototypage avancé réalisée conjointement avec le client. Il permet de construire le système avec l'utilisateur, en étant totalement basé sur des prototypes successifs. RAD préconise d'organiser la participation des utilisateurs tout au long de la conception en leur fournissant des supports à leur prise de décision. La phase de prototypage débouche sur une interface validée par le client. De nombreuses organisations ont employé ce type de développement dans les années 90 et ont eu des soucis lors de la maintenance des applications ainsi développées à cause du manque de conception inhérent à la démarche. La conception calquée sur l'interface complique la lecture d'un programme car la logique métier est distribuée dans l'ensemble de l'outil. Cela complique la correction des problèmes éventuels.

L'approche itérative RAD pose trois exigences majeures et difficiles à satisfaire :

a. La grande implication des utilisateurs : Difficile à obtenir au même degré pour l'ensemble des utilisateurs ;

b. Une équipe experte capable de s'adapter et de répondre rapidement aux attentes du client ;
c. Un atelier de génie logiciel résistant permettant d'obtenir rapidement un prototype à partir du concept.

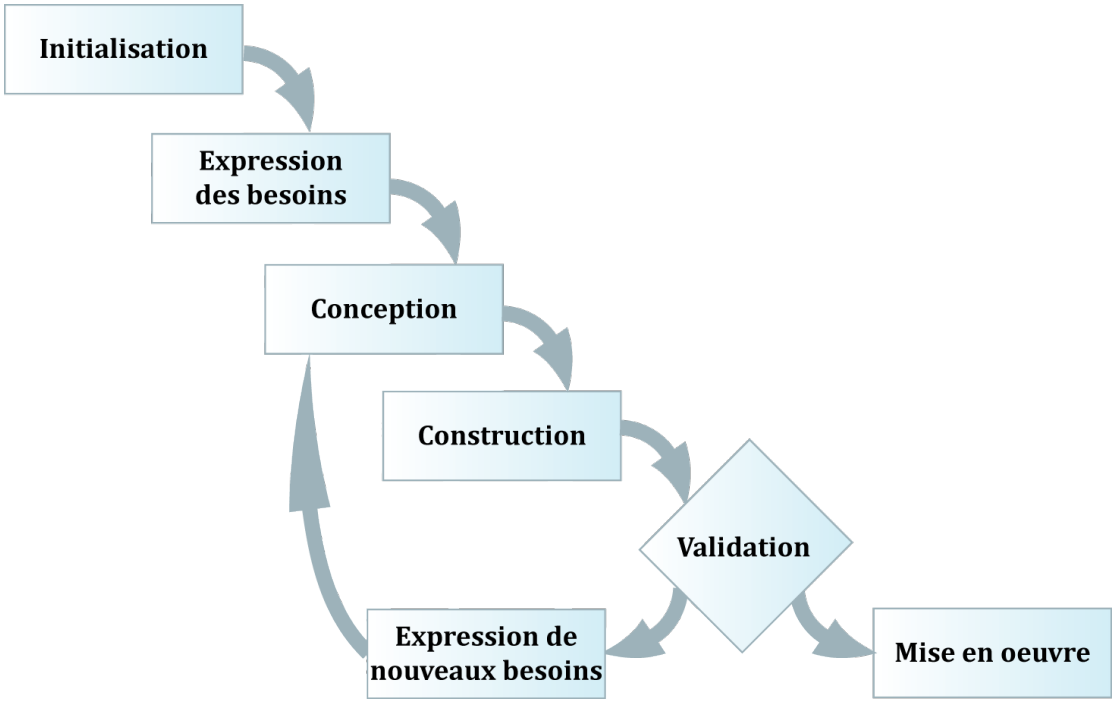


Figure 0-4 : Le cycle RAD

Le succès des langages orientés objet a donné naissance à de nombreuses méthodologies de conception associées à ce mode de développement de logiciels [AND 98]. Toutefois, la réunion de trois principales approches, portées respectivement par I. Jacobson (OOSE), J. Rumbaugh (OMT) et G. Booch (Booch'93) et la naissance de la notation UML (« Unified Modeling Language ») a fédéré en grande partie la communauté de conception logicielle orientée objet [GAB 98]. Suite à cette normalisation, le processus « Unified Process » (UP) est venu soutenir la systémique du méta-modèle de la notation UML. Cette méthode est générique et doit donc être adaptée à chacun des projets. Elle présente plusieurs caractéristiques essentielles :

a. Pilotage par les cas d'utilisation : Les cas d'utilisation sont les véritables pilotes du projet. Ils sont utiles à la définition des besoins en terme de fonctionnalités mais également pour l'analyse ou le test.

b. Méthode itérative et incrémentale : Comme les méthodes présentées ci-dessus, UP se base sur un développement incrémental avec une validation des utilisateurs régulière.
c. Gestion des besoins et des exigences : UP recommande une gestion disciplinée des besoins et des exigences du projet.
d. Modélisation : UP préconise la modélisation visuelle, et plus particulièrement en utilisant les différents diagrammes présents dans la notation UML, pour favoriser la communication.

5. Synthèse

L'inconvénient majeur des méthodes traditionnelles de conception logicielle, telles que le cycle en V ou en cascade, est de ne pouvoir facilement identifier des erreurs au moment de la définition des besoins. De plus, l'utilisateur est généralement écarté dans le processus de développement dès la phase de conception. La validation suivante ne se fait qu'à la livraison du logiciel. Pour palier ce problème, le modèle en spirale choisit de proposer une méthode implémentant des versions successives d'un logiciel en reprenant les différentes étapes du cycle en V. Le prototypage successif favorise, alors, l'expression progressive des besoins tout au long des étapes de conception. Ce modèle reste relativement lourd et implique donc des projets coûteux et longs à conclure. Un besoin important d'allègement des méthodes de conception logicielle a favorisé l'émergence des méthodes dites agiles. De manière générale, celles-ci ont pour but d'augmenter le niveau de satisfaction des clients tout en rendant le travail de développement plus facile. Afin d'obtenir des résultats intéressants, ces méthodes demandent une très forte implication des utilisateurs avec un potentiel de décision relativement important, ce qu'il est difficile d'obtenir dans chacun des projets. Il est toutefois peu recommandé de se diriger vers l'autre extrême, c'est-à-dire le cycle de vie des logiciels libres. En effet, celui-ci impose une livraison très fréquente : aussitôt qu'une fonctionnalité est disponible, le produit est rendu public. Ceci explique pourquoi bon nombre de produits ne sont pas matures et ni fiables à leur première distribution.

Dans la suite, nous présenterons la norme ISO 13407 visant à supporter méthodologiquement un processus de conception de système centré sur la satisfaction des besoins utilisateur.

B. Approche de conception centrée utilisateur

La conception centrée utilisateur (CCU) est une méthode de conception, ayant pour objectif de rendre les systèmes utilisables. Pour atteindre cet objectif, cette méthode fait intervenir l'utilisateur tout au long du processus de conception. Cette approche est normalisée par la norme ISO 13407 [ISO 99].

Selon cette norme, la CCU, utilisée dans le cas de systèmes interactifs, permet de réaliser des gains notables sur le plan socio-économique. L'intervention des utilisateurs finaux pendant les étapes de conception et de validation permettra de proposer des solutions en meilleure adéquation avec leurs besoins. Accroître l'utilisabilité des outils produits permettra aux utilisateurs finaux d'atteindre plus efficacement leurs objectifs. Concevoir une application performante selon la CCU nécessite donc de se demander à chaque étape critique de la conception si le produit correspond aux besoins des utilisateurs finaux.

1. La norme ISO 13407

La norme ISO 13407 ne propose pas une démarche unique de conception pour les systèmes interactifs mais détermine les exigences auxquelles une méthodologie doit répondre pour être considérée comme « centrée utilisateur ». En effet, elle ne couvre pas l'ensemble des activités nécessaires de conception. Elle est donc complémentaire des méthodes de conception existantes. Cette norme a ensuite été révisée sous la référence ISO 9241-210 [ISO 10].

Selon la norme ISO 13407, l'intégration d'une approche centrée sur l'utilisateur se caractérise par :

a. La participation active des utilisateurs et une compréhension claire des exigences liées à l'utilisateur et à la tâche : Cette participation représente une source non négligeable de connaissances sur le contexte d'utilisation, les tâches et la manière dont les utilisateurs seront amenés à travailler;

b. Une répartition appropriée des fonctions entre les utilisateurs et la technologie : Les décisions de conception visent à déterminer dans quelle mesure une tâche doit faire l'objet d'un traitement automatisé ou être assignée à un opérateur humain;
c. L'itération des solutions de conception : L'itération, conjuguée à une implication active de l'utilisateur, est un moyen efficace pour minimiser les risques de concevoir un système en inadéquation avec les exigences liées à l'utilisateur et à l'organisation;
d. Une conception pluridisciplinaire : La CCU fait appel à une grande variété de compétences.

2. Étapes du processus de conception centrée utilisateur

La norme ISO 13407 définit le processus de conception comme présenté sur la Figure 0-5. Le processus est d'abord planifié, c'est-à-dire que nous cherchons à spécifier comment les activités centrées sur l'opérateur humain vont s'intégrer dans le processus global de développement. La première étape du cycle consiste à définir le contexte d'utilisation, suivie de l'étape d'acquisition des besoins utilisateur. L'équipe doit ensuite être capable de proposer des solutions de conception qui feront l'objet d'évaluation.

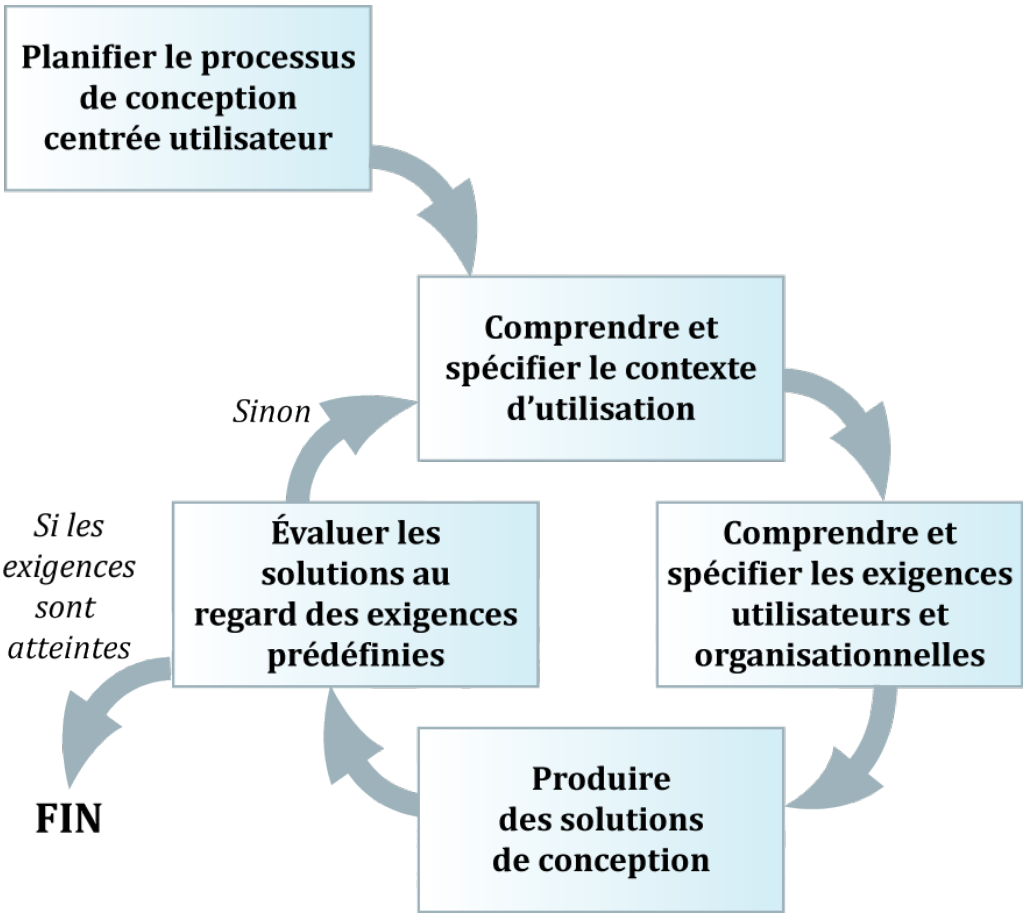


Figure 0-5 : Processus de conception centrée utilisateur

a) Planification du processus de conception

Le pré-requis essentiel à une démarche CCU consiste à planifier sa mise en œuvre et donc son intégration à un processus de conception de système existant tant sur les plans techniques que sur la conduite de projet.

Ce plan devra permettre d'identifier clairement les étapes de conceptions liées à la démarche CCU et les procédures permettant d'intégrer ces activités à d'autres activités de développement de systèmes. Chacune de ces activités devra ensuite être placée sous la responsabilité d'individu ou d'organisme. L'adaptation des outils et des méthodes se basera sur la consultation de documentations et des discussions autour des pratiques et des contraintes de la conception. Ces discussions sont notamment menées avec la participation des futurs utilisateurs du système.

b) Spécification du contexte d'utilisation

La première étape, à proprement parler, du cycle de CCU vise à comprendre et spécifier le contexte d'utilisation. Le contexte dans lequel le système sera utilisé est défini par les caractéristiques des utilisateurs, des tâches et des environnements organisationnels. La définition de ce contexte d'utilisation permettra de guider les premières décisions de conception.

L'identification des profils d'utilisateur potentiel représente une tâche essentielle pour bien initier cette démarche. La connaissance de ces profils permettra de choisir les méthodes d'évaluation et ainsi définir les profils nécessaires à l'évaluation des différentes solutions de conception proposées. Les caractéristiques propres aux utilisateurs peuvent inclure les connaissances, les compétences, l'expérience, l'éducation, la formation…

Il faudra également définir les tâches confiées aux utilisateurs, et plus particulièrement les caractéristiques des tâches capables d'influencer l'utilisabilité telles que la fréquence et la durée d'exécution.

Des méthodes fondées sur l'observation et le recours aux questionnaires sont proposées dans la norme. Dans le cas de certains projets, une analyse de l'existant sera nécessaire. Dans le domaine des interactions homme-machine, l'analyse des tâches utilisateur permet de comprendre comment le système sera utilisé, identifier les tâches critiques, les plus fréquentes, leur importance et leurs niveaux de difficulté. Le choix des méthodes à utiliser est fonction du contexte technique, de la disponibilité des différents acteurs.

c) Spécification des besoins utilisateur

L'étape suivante de conception consiste à définir précisément les exigences liées à l'utilisateur et à l'organisation, par rapport au contexte d'utilisation préalablement défini. Chaque exigence devra spécifier le profil d'utilisateur potentiel et les personnes impliquées dans sa conception. Chacune des exigences sera ensuite priorisée et des critères mesurables permettant le diagnostic leurs seront adjointes.

À ce niveau, les objectifs d'utilisabilité vont guider et justifier les choix de conception. Ils fourniront par la suite des critères de validation lors des tests utilisateurs.

d) Production des solutions

L'équipe de conception doit être capable, à la suite des deux étapes précédentes, de proposer des solutions de conception. Celles-ci sont produites en fonction de l'état de la technique, l'expérience et les connaissances des participants, ainsi que les résultats de l'analyse du contexte d'utilisation.

L'objectif principal de cette étape est donc de matérialiser davantage les solutions de conception à l'aide de maquettes, prototypes, etc. en offrant la possibilité aux utilisateurs d'exécuter des tâches.

e) Evaluation des solutions

L'étape de validation consiste à évaluer l'adéquation entre les solutions proposées par l'étape précédente et les exigences liées aux utilisateurs et à l'organisation. Celle-ci constitue donc une étape essentielle car elle permet de fournir un retour d'information notable pouvant contribuer à améliorer la solution proposée ou de juger si les objectifs ont été atteints. L'accent sera principalement mis sur le retour d'information pendant les premières phases du projet afin d'orienter la conception. Lorsque le prototype plus complet est disponible, il est possible de déterminer dans quelle mesure les objectifs sont atteints.

Le pilotage des tests utilisateur selon un protocole d'évaluation précis permet de détecter les défauts, que nous ordonnons en fonction des objectifs d'utilisabilité définis précédemment. Diverses méthodes sont proposées dans la norme telles que les tests d'évaluation individuels ou en groupe et/ou les questionnaires de satisfaction.

f) Synthèse

Cette norme présente une démarche de conception centrée utilisateur pouvant s'adapter à la conception ou à la refonte de tout système informatique. Elle ne présente pas un cadre méthodologique précis et rigide mais préconise les étapes nécessaires à incorporer dans un processus de conception traditionnel.

Il paraît relativement difficile d'appliquer cette démarche à des situations réelles de conception. En effet, les fortes contraintes pesant sur un projet de développement, les pratiques de conception habituelles et les possibilités technologiques peuvent rendre difficile l'application stricte d'une démarche centrée sur l'utilisateur. Il s'agit alors de garder les lignes directrices de cette démarche, c'est-à-dire le cycle itératif de conception et la validation par un retour d'information de la part des utilisateurs à chaque itération.

Pour rendre applicable cette norme, il est donc primordial d'alléger cette démarche en simplifiant et en automatisant les interactions entre les différents participants du projet. Dans l'idéal, la modification du contexte d'utilisation devrait permettre de faire évoluer quasi instantanément les solutions de conception. Malheureusement, ce type d'automatisme ne peut s'appliquer pour tous les types d'exigence. Le fait de se concentrer sur les exigences fonctionnelles permettrait d'optimiser le processus de conception.

C. Techniques de recueil et d'analyse des besoins

Pour que la rencontre entre l'utilisateur et le concepteur aboutisse à un système utilisable, des moyens de compréhension mutuelle sont nécessaires. Nous définissons dans cette partie ces différents moyens qui permettent de mieux identifier, produire et formaliser les besoins de l'utilisateur [BRAN 03].

1. Les méthodes d'analyse de l'usage

L'analyse des usages vise à recueillir des informations sur la manière dont les personnes utilisent des interfaces, qu'elles proviennent de solutions existantes ou de prototypes, en vue de les améliorer ou de concevoir de nouveaux produits. Ces méthodes s'inspirent de la méthode ethnographique[2]. L'analyste joue le rôle d'observateur chargé de recenser le comportement des utilisateurs et leur contexte d'utilisation. Ce type de méthode implique obligatoirement d'aller sur le terrain afin de rencontrer les utilisateurs cibles et de mettre en œuvre plusieurs méthodes de ce type telles que l'entretien, le questionnaire, l'observation...

L'entretien

L'entretien est une technique de recueil des besoins où nous demandons à l'utilisateur de s'exprimer sur ses besoins et ses difficultés à propos de l'utilisabilité d'un système [BLA 10]. Celle-ci est une technique fondamentale de la plupart des processus d'identification des besoins en génie logiciel. Lorsqu'il est bien conduit, l'entretien constitue une méthode efficace pour récolter à la fois des données sur les utilisateurs, leurs usages, l'utilisabilité perçu et/ou voulu d'un système. L'un des freins à une bonne évaluation des besoins recueillis par entretien est la complexité d'analyse des données en l'absence de méthodologies structurantes [KOT 98] [SOM 97].

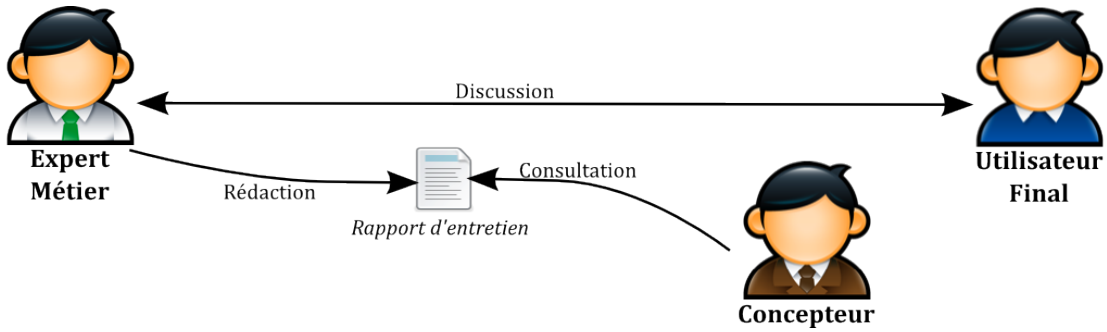


Figure 0-6 : Schéma descriptif de la méthode par entretien

[2] L'ethnographie a pour objectif l'identification des pratiques, des mythes et des croyances, composantes de la culture, d'une population donnée. Elle cherche à décrire les règles d'interactions sociales dans un contexte particulier par l'observation participative.

Le questionnaire

Une technique, utilisée généralement en complément des entretiens, consiste à collecter les données sur les usages à l'aide de questionnaires [SIN 08]. Il est structuré sous la forme d'une liste de questions ouvertes ou fermées méthodiquement posées. Cette méthodologie peut trouver des applications intéressantes lorsque nous disposons d'un échantillon important et représentatif des utilisateurs, lorsque nous souhaitons mesurer la satisfaction des utilisateurs. Des problèmes d'échantillonnage, de représentativité et de généralisation des résultats peuvent être rencontrés avec cette technique. De plus, une investigation de ce type ne permet pas de préconiser des améliorations mais seulement de définir un état de l'opinion des utilisateurs.

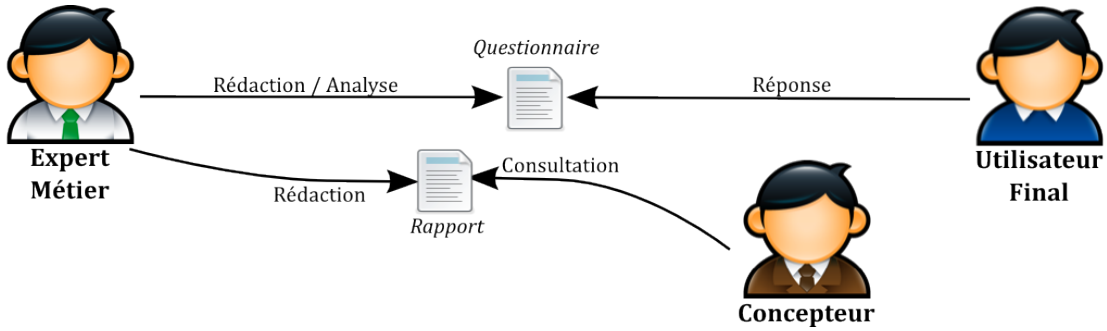


Figure 0-7 : Schéma descriptif de la méthode par questionnaire

L'observation

Alors que les entretiens et les questionnaires fournissent des informations sur un usage, ils ne permettent pas de cerner l'activité réelle. En effet, l'observation permet d'identifier réellement ce que fait l'utilisateur car celle-ci n'est pas basée sur l'interprétation faite par l'acteur lui-même [ARB 10]. Par ailleurs, l'entretien et le questionnaire impliquent toujours un décalage dans le temps, qui implique une vision déformée des connaissances. A l'inverse, l'observation implique que l'utilisateur accomplisse réellement son activité. Elle présente l'avantage d'annuler les inconvénients liés aux techniques d'évocation explicite des faits, comme c'est le cas dans un entretien où l'utilisateur décrit ce qu'il croit être sa connaissance d'un usage.

Le sociologue américain, I. Deutcher, présente dans une de ses études [DEU 74] un certain nombre de résultats contradictoires obtenus selon que l'enquête est conduite par observation ou par questionnaire. Ces différences sont expliquées par la relation directe et particulière avec les individus qui induit des comportements pouvant être opposés aux attitudes exprimées.

L'observation peut être complété par des techniques de verbalisation, qui consiste à demander à l'utilisateur de réaliser son activité en décrivant son activité à « voix haute ». Une alternative peut être proposée : la verbalisation consécutive qui consiste à demander à l'utilisateur de commenter un film de sa propre activité.

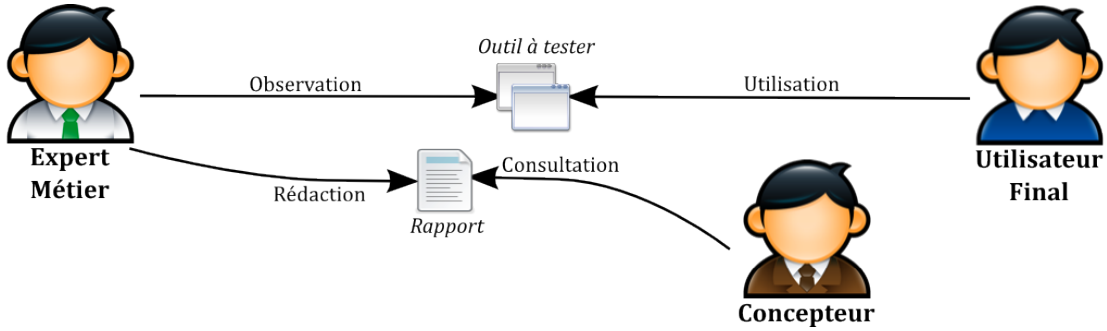


Figure 0-8 : Schéma descriptif de la méthode par observation

Traces de l'activité

Une autre technique de recueil consiste à analyser les traces de l'activité de l'utilisateur, encore appelée « monitoring » [MIL 06]. Cette technique permet de quantifier les durées et fréquences d'utilisation, les fonctionnalités utilisées, les erreurs rencontrées ou encore les opérations et abandons réalisés. Cette technique est riche mais présente deux inconvénients majeurs : la déontologie obligeant l'analyste à demander l'accord des utilisateurs et la difficulté d'interprétation des résultats car nous ne connaissons que ce qu'a fait l'utilisateur mais pas ce qu'il souhaitait faire.

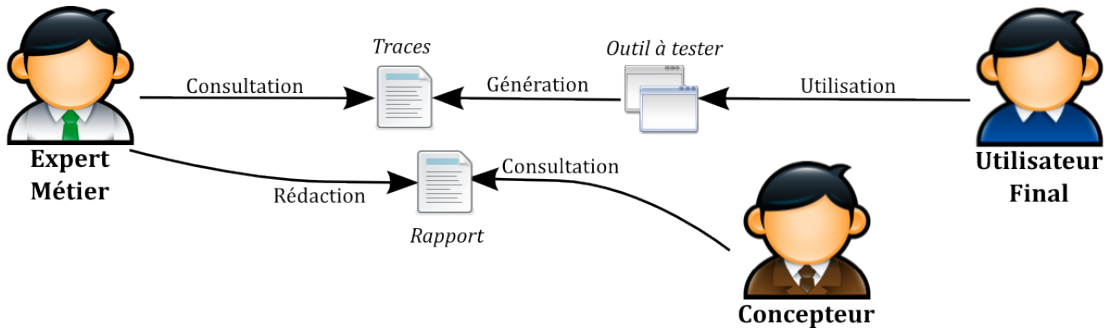


Figure 0-9 : Schéma descriptif de la méthode par analyse des traces

2. Les méthodes d'inspection de l'utilisabilité

Les méthodes de l'inspection de l'utilisabilité consistent à évaluer les capacités d'un outil à être efficace, efficient, tolérant aux erreurs, facile à apprendre et satisfaisant pour ses utilisateurs. Celles-ci sont réalisées par des spécialistes de l'utilisabilité en se référant soit aux connaissances de l'évaluateur, soit à des outils.

Inspection heuristique

Lors d'une inspection heuristique, un expert examine une interface pour identifier le plus grand nombre de difficultés [NIE 90]. Pour ce faire, il s'appuie sur une grille d'évaluation contenant des critères ergonomiques tels que la lisibilité, la brièveté… L'inspection heuristique repose donc sur la pertinence des grilles mais aussi sur le niveau d'expertise de l'inspecteur. Afin d'obtenir une inspection fine et détaillée, il est préconisé de disposer de plusieurs experts.

Évaluation collaborative

L'inspection heuristique est menée sans avoir une connaissance précise des besoins et des tâches des utilisateurs finaux. Pour palier ce problème, l'évaluation collaborative propose de réaliser l'évaluation en partenariat avec un utilisateur réel [MUL 98]. Celui-ci est invité à décrire oralement ses tâches. Ses commentaires viendront ensuite compléter le rapport d'inspection.

3. Les méthodes expérimentales

Certaines méthodes cherchent à mesurer expérimentalement l'utilisabilité d'un outil, ce qui se déroule dans un laboratoire d'usage spécialement dédié [BAS 91].

Un test est une situation où nous installons un échantillon d'utilisateurs cibles dans une pièce où ils se savent observés. Les résultats des tests sont ensuite étudiés de manière quantitative et qualitative et enfin interprétés par les évaluateurs afin de proposer des améliorations de l'outil. Différents profils peuvent prendre part à l'évaluation afin de définir rapidement les recommandations adaptées.

Ce type de test présente l'avantage d'être riche en information, formalisé, efficace et rapide. De plus, le contexte du test permet de créer un espace de dialogue où évaluateurs, concepteurs et utilisateurs peuvent définir des pistes communes pour améliorer l'utilisabilité.

4. Les méthodes participatives et créatives

A l'opposé des méthodes rigoureuses, telles que les méthodes expérimentales, il est possible d'utiliser des méthodes plus souples basées sur la communication entre les acteurs des projets technologiques. Elles sont généralement utilisées en complément des méthodes d'analyse de l'usage.

Les groupes de discussion

Les groupes de discussion consistent à regrouper une dizaine d'utilisateurs encadrée par un psychologue. Les utilisateurs sont invités à s'exprimer librement sur l'outil, leurs besoins, leurs insatisfactions… L'objectif de ces groupes est de produire des verbalisations sur la manière dont les utilisateurs s'échangent des informations sur l'outil.

Scénario

La méthode des scénarios [ROS 02] consiste à évaluer la réaction d'utilisateurs d'un système qui n'existe pas encore en leur présentant des scénarios sous forme de récits, de dessins ou de photos. Les réactions obtenues sont ensuite considérées comme des indicateurs de l'utilité et de l'acceptabilité du système.

Les méthodes participatives

Les méthodes participatives consistent à impliquer les utilisateurs dès la conception de l'outil. L'utilisateur n'est pas une simple source d'information mais

un acteur de la conception du système d'information. L'idée principale est que la participation directe de l'utilisateur augmente l'acceptation du changement. Ce type de méthodes renvoie aux modalités de conduite de projet [BRAN 04], à la méthode de conception centrée sur l'utilisateur ainsi qu'aux méthodes agiles de conception définies ci-dessus.

5. Synthèse

Pour réaliser ses buts, l'utilisateur doit d'abord avoir à sa disposition un certain nombre de fonctionnalités. Ensuite, l'utilisateur souhaite bénéficier d'outils efficients, c'est-à-dire permettant de l'aider dans la réalisation de ses tâches. L'utilisation d'un logiciel doit non seulement permettre d'atteindre le but fixé (efficacité), mais également être efficient. Tandis que les méthodes d'analyse de l'usage pourront être utilisées au début d'un projet pour identifier les besoins initiaux, les méthodes expérimentales seront plus utiles dans les dernières phases du projet pour prouver l'utilisabilité de l'outil. Les méthodes d'inspection, plus coûteuse, seront elles aussi utilisées lors des validations finales. Pour conclure, les méthodes participatives et créatives présentent beaucoup d'analogies avec les méthodes centrées utilisateur et donc les méthodes agiles de conception. Celles-ci sont, avec les méthodes d'analyse de l'usage, les méthodes les plus utilisées.

Méthode	Participant	Communication directe	Moyens nécessaires
Entretien	Expert métier Utilisateur final	✓	
Questionnaire	Expert métier Utilisateur final		Questionnaire
Observation	Expert métier Utilisateur final	✓	Outil à tester
Analyse des traces de l'activité	Expert métier Utilisateur final		Outil à tester
Inspection	Expert en interface		Outil à tester

heuristique			
Évaluation collaborative	Expert en interface Utilisateur final	✓	Outil à tester
Méthodes expérimentales	Utilisateur final Évaluateur		Laboratoire d'expérimentation
Groupes de discussion	Utilisateur final Psychologue	✓	
Scénario	Utilisateur final		Scénario
Méthodes participatives	Utilisateur final Concepteur	✓	

Tableau 0-1 : Synthèse des techniques de recueil des besoins

Comme nous pouvons le voir sur le Tableau 0-2, les méthodes présentées ne font pas intervenir les mêmes catégories de participants. L'utilisateur final est impliqué dans l'ensemble des méthodes hormis l'inspection heuristique. Cette méthode ne cherche qu'à évaluer la qualité des interfaces sans aucune analyse métier. L'évaluation collaborative complète cette méthode en apportant une expertise métier supplémentaire. Il est intéressant de voir que les différentes méthodes ne font pas toutes appel à un contact direct avec l'utilisateur final. Les méthodes d'analyse des traces ou de questionnaire font référence à des résultats produits par une activité demandée à l'utilisateur final. Ces résultats peuvent donc être altérés par un point de vue personnel de chaque participant. L'usage combiné de ces différentes méthodes peut produire des résultats fiables permettant une conception aisée. Nous pouvons noter que les méthodes participatives présentées dans le tableau font référence à des méthodologies de conception agiles dans lequel l'utilisateur est fortement impliqué.

D. Méthodologies en ingénierie des besoins

La principale mesure de succès d'un SII est le degré de superposition entre ce système et son but. Par conséquent, l'identification de ces buts doit être une des activités principales dans le processus de conception d'un SII. Il est depuis

longtemps reconnu qu'une définition des besoins inadéquate, incomplète, ambigüe ou inconsistante a un impact non négligeable sur la qualité du système produit. Ainsi, l'ingénierie des besoins, une branche de l'ingénierie logicielle, a pour objectif de capturer, raffiner, définir les besoins d'un logiciel.

L'ingénierie des besoins va permettre de mieux identifier, comprendre, valider et interpréter les besoins des utilisateurs afin de réaliser le système nécessaire et adéquat. Elle a été initialement décrite comme : « la partie du développement au cours de laquelle les gens tentent de découvrir ce qui est désiré » [GAU 89]. C'est pour cette raison que les premières méthodes se sont concentrées sur la définition du système donc ce qui devait être réalisé : le « quoi ? ». Puis elles se sont concentrées sur comment le système devrait être défini, organisé puis implanté : le « comment ? ». Ces différentes définitions restaient très techniques et inaccessibles à des opérationnels.

Afin d'impliquer les différents profils opérationnels, les approches se sont attachées aux raisons pour lesquelles le système est nécessaire : le « pourquoi ? ». Dans celles-ci, le système désiré est vu comme un moyen d'atteindre les différents objectifs de l'organisation afin de répondre à ses problèmes organisationnels. La définition des besoins ne définit plus le système à réaliser mais les activités supportées par le système. Deux grandes familles sont proposées par la communauté scientifique pour identifier les besoins. La première est centrée autour de la notion de scénario tandis que la seconde autour de la notion de but.

Ingénierie basée sur les scénarios

Comme le décrit A. Sutcliffe [SUT 03], un scénario est une représentation du monde réel. Dans le cas de l'ingénierie des besoins, un scénario est limité à la compréhension du système par les participants et se situe au niveau d'abstraction du système (cf. Figure 0-10). Les évènements sont des interactions entre l'utilisateur et le système. Il existe deux grandes méthodes proposées dans cette approche : la méthode ScenIC [POT 99] et la méthode SCRAM [SUT 98].

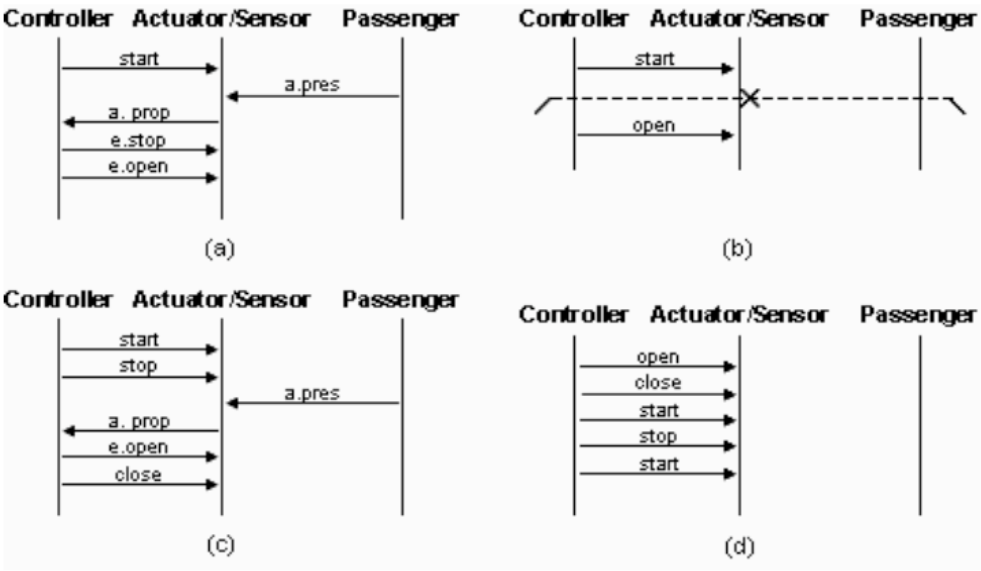


Figure 0-10 : Exemple de scénarios utilisés [DAM 06]

Ingénierie orientée par les buts

Comme le définit A. Lamsweerde [LAM 01], un but doit être atteint par le système. Il peut donc être fonctionnel, c'est-à-dire qu'il représente un service fourni, ou non fonctionnel, c'est-à-dire qu'il représente une contrainte telle que le niveau de sécurité. Dans la majorité des cas, les buts sont généralement fixés implicitement par les participants. À partir du moment où l'analyse préliminaire des besoins a été faite, d'autres buts peuvent être obtenus par raffinement, abstraction… Différentes méthodes d'ingénierie orientée par les buts seront présentées dans la suite.

Comparaison

Les buts se concentrent sur l'abstraction des besoins décrits de manière ambiguë par les utilisateurs. Les scénarios permettent de compléter ces buts en précisant comment le système va fonctionner pour répondre à la demande. Les langages orientés par les buts sont généralement plus simples et requièrent moins d'efforts en comparaison des techniques basées sur les scénarios. La mise en place d'une méthodologie à l'aide des scénarios nécessite plus de temps et de ressources qu'une approche par les buts. Pour conclure, les scénarios permettent de préciser les buts.

Malgré que langages en ingénierie orientée par les buts soient plus simples, ils restent généralement complexes à mettre en œuvre et surtout à comprendre [MIS 05]. Ils restent illisibles pour une personne non experte dans ces méthodologies. Les difficultés de prise en main et de compréhension sont de véritables verrous à une bonne utilisation de ces méthodologies basées sur les buts. Nous allons le voir dans la présentation des principales méthodes orientées par les buts dans la suite de ce chapitre.

1. KAOS

La méthodologie KAOS (« Knowledge Acquisition in autOmated Specification » [DAR 93] ou « Keep All Objects Satisfied » [LAM 03]) est une approche d'ingénierie des exigences orientée par les objectifs avec un ensemble important de techniques d'analyse formelle. KAOS peut être décrit comme un outil multi-paradigme permettant de combiner différents niveaux d'expression : semi-formel pour la modélisation et la structuration des buts, qualitatif pour la sélection entre les différentes alternatives et formel pour l'ajout de précisions dans certains cas.

KAOS est une méthode qui préconise une phase de modélisation des exigences avant d'écrire le cahier des charges proprement dit. Le cahier des charges devient alors un produit dérivé qui dressera un inventaire structuré et motivé des exigences sur la base du modèle construit.

Modèle

Un modèle KAOS fédère 4 vues complémentaires et liées sur le système d'information :

a. Le modèle des buts permettant de représenter la hiérarchie des buts ;
b. Le modèle de responsabilité permettant de décrire pour chaque agent ses exigences et ses attentes ;
c. Le modèle objet pour représenter les différentes entités ;
d. Le modèle des opérations pour définir les différents services fournis aux agents.

Dans un modèle KAOS, nous trouvons donc des buts (*Goal*), des exigences sur le système informatique (*Requirement*), des attentes sur l'environnement de ce système (*Expectation*), des conflits entre objectifs, des obstacles (*Obstacle*), des entités, des agents, etc. La Figure 0-11 présente une version minimale du méta-modèle KAOS. Il est possible de définir avec ce méta-modèle un arbre de décomposition en utilisant des objectifs, des attentes, des exigences ainsi que des propriétés du domaine. La notion d'agent permet de définir des responsabilités mais seulement sur les exigences et les attentes. Des modèles plus précis peuvent être consultés dans les annexes (cf. Annexe A).

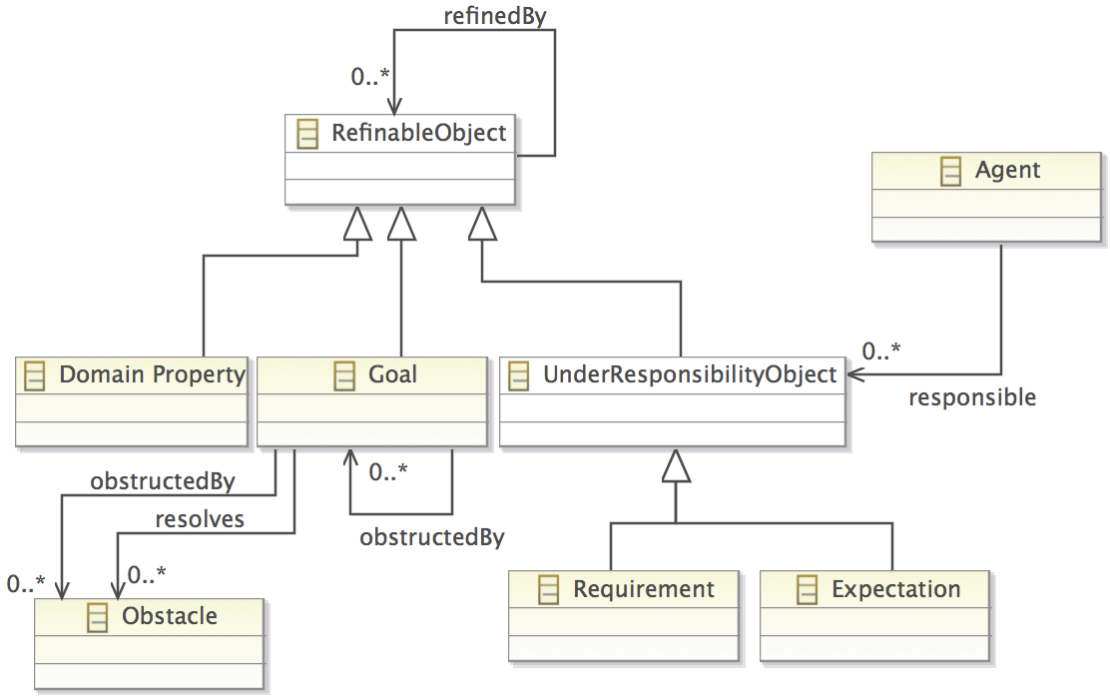


Figure 0-11 : Méta-modèle KAOS

2. i*

i* [YU 97] est un langage de modélisation, orienté agent, adapté à la phase préliminaire de modélisation du système afin de comprendre les problèmes du domaine. C'est une approche développée initialement pour la modélisation d'organisation et leurs systèmes d'informations composés d'acteurs hétérogènes avec des objectifs différents. Cette méthode permet de modéliser le « qui » et le « pourquoi » mais pas le « quoi ».

Méta-modèle

Le méta-modèle de la méthodologie i* permet de définir des liens entre objectifs. Ils peuvent se décomposer en quatre éléments : but (*HardGoal*), objectif (*SoftGoal*), tâche (*PersonalGoal*) sous la responsabilité d'un agent (*Actor*). La Figure 0-12 présente une version minimale du méta-modèle i*. Il est important de définir la différence entre *HardGoal* et *SoftGoal* : un *HardGoal* est un objectif avec un critère de satisfaction logique et claire tandis que le critère d'un *SoftGoal* est plus flou et donc plus difficile à vérifier. Il est également possible de définir des obstacles (*Obstacle*) qui peuvent empêcher l'achèvement d'un objectif ou encore être contourné grâce à un objectif.

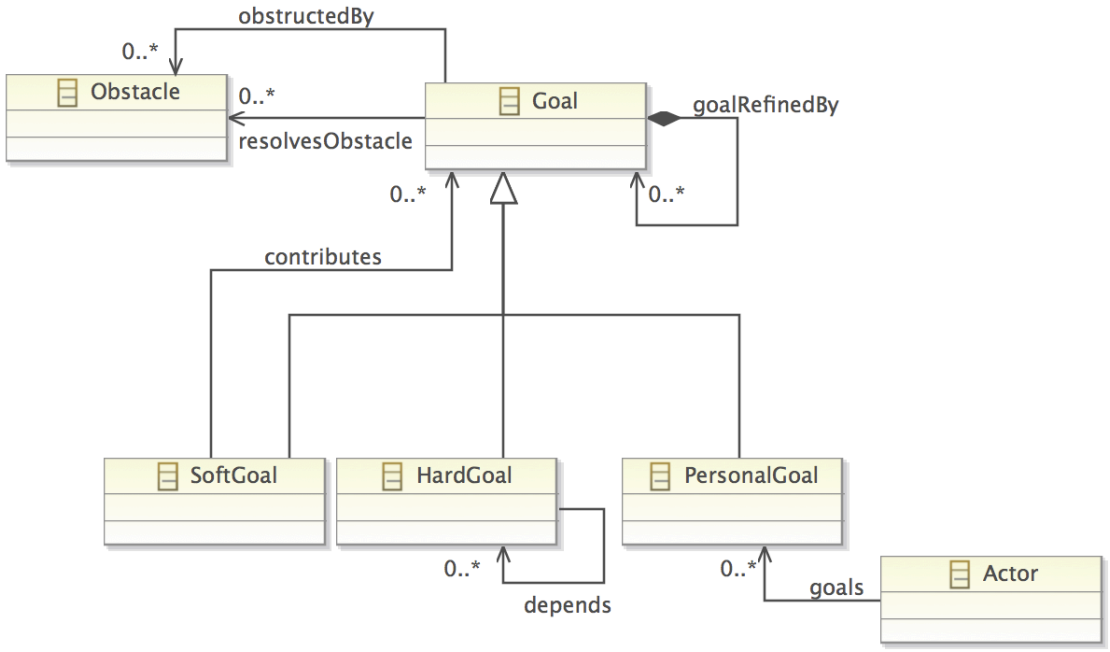


Figure 0-12 : Méta-modèle i*

Modèle

La méthodologie i* propose deux principaux modèles : le modèle « Strategic Dependency » (SD) et le modèle « Strategic Rationale » (SR).

Un modèle SD est un graphe où chaque nœud représente un acteur et chaque lien entre acteurs une dépendance. Le modèle SD est utilisé pour exprimer les relations de dépendances entre acteurs dans un contexte organisationnel. Ce modèle permet d'identifier les acteurs et qui dépend du travail d'un acteur.

Un modèle SR est utilisé pour explorer les raisons associées à chaque acteur et leurs dépendances. Il permet également de définir des informations sur les moyens utilisés par un acteur pour atteindre un but ou un objectif. Le modèle SR permet de visualiser les dépendances entre acteurs en incluant le modèle SD mais en fournissant un niveau de modélisation plus précis. Il est possible de naviguer à l'intérieur d'un acteur.

3. Tropos

La méthodologie i* est utilisée comme base pour la méthodologie Tropos [CAS 02] en y ajoutant un langage de spécification formel, appelé « Formal Tropos », permettant d'ajouter des contraintes, invariants, pre- et post-conditions.

Tropos utilise principalement i* pour modéliser les systèmes. Ce langage se focalise sur l'ingénierie des besoins centrée sur les caractéristiques intentionnelles des agents comme présentée ci-dessus. Bien que i* propose un langage graphique, les diagrammes qui modélisent de réelles applications complexes peuvent vite devenir illisibles. Par ailleurs, i* ne peut être utilisé que dans la phase d'analyse puisqu'il permet uniquement d'exprimer les besoins. Pour cette raison, Tropos a recours à l'utilisation d'autres notations inspirées d'UML mais aussi du langage de spécification formel proposé par la méthodologie.

Tropos couvre une grande partie du cycle de développement. Elle suit une approche incrémentale par raffinement itératif. La démarche proposée par Tropos se décompose en cinq phases [GIU 02] :

a. L'analyse des besoins initiaux permet l'identification des principaux acteurs et de leurs buts respectifs ;
b. L'analyse des besoins finaux est un raffinement des modèles précédents en fournissant une description du système dans son environnement ;
c. La conception architecturale permet de définir l'architecture du système décomposé en sous-système représenté par des acteurs ;

d. La conception détaillée permet d'approfondir le modèle précédent en spécifiant les entrées-sorties de chacun de ces composants ;
e. L'implantation finale.

Tropos est une méthodologie générique pouvant être appliquée à des contextes différents. Elle couvre la totalité du cycle de développement, et se base sur une notation formelle qui lui permet de vérifier et valider ses modèles. Par contre, Tropos ne couvre qu'un ensemble limité de concepts orientés agent. La dimension organisationnelle, par exemple, n'est pas prise en compte de manière explicite.

4. NFR

La méthodologie NFR (« Non-Functional Requirements »), proposée dans [MYL 92] puis approfondie dans [CHU 00], se concentre, comme son non l'indique, sur la modélisation des exigences non fonctionnelles. Il est possible de définir trois types de buts :

a. *SoftGoal* représentant les exigences non fonctionnelles à considérer ;
b. *Operationalization* représentant les techniques de bas niveau utiles à la satisfaction des *SoftGoal* ;
c. *Claim* (non présent sur le méta-modèle) permettant à l'analyste d'enregistrer les caractéristiques de chaque *SoftGoal* , telles que la priorité ou la raison.

Il est ensuite possible de définir plusieurs relations entre ces objectifs : la décomposition à l'aide d'opérateur AND/OR et la contribution positive ou négative. La Figure 0-13 présente une version minimale du méta-modèle NFR.

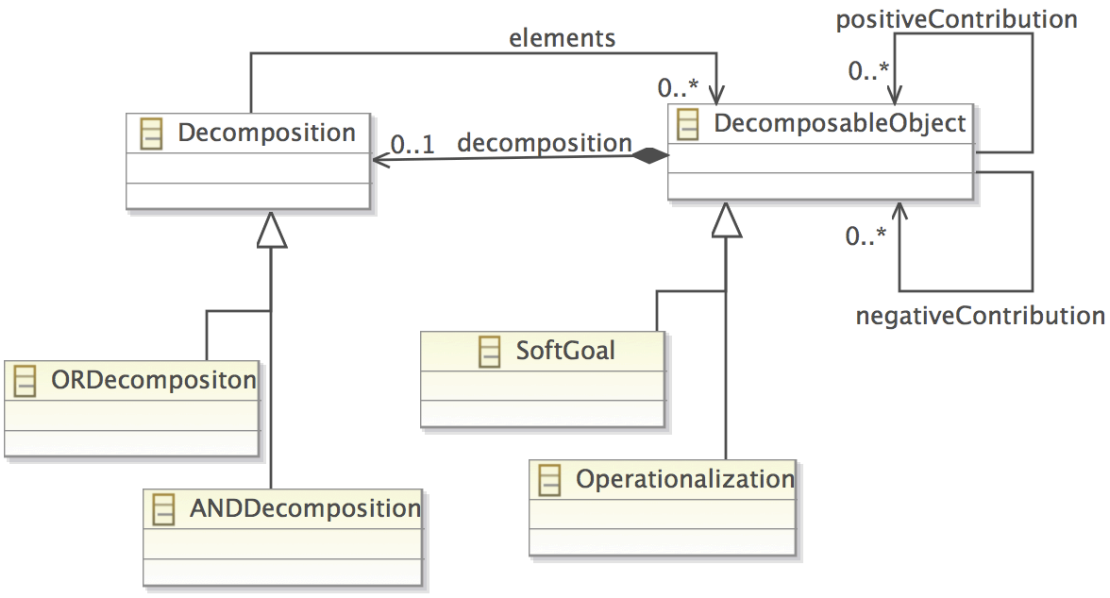


Figure 0-13 : Méta-modèle NFR

5. Synthèse

L'ingénierie des besoins n'est pas seulement un processus de découverte et de spécification des exigences, c'est aussi un processus pour faciliter la communication, la diffusion et la compréhension de ces besoins entre les différents participants. Les méthodologies présentées ci-dessus paraissent complexes au premier abord. En effet, la majorité des méthodes contient plusieurs modèles manipulant une quantité relativement importante de concepts manipulables. Ces inconvénients augmentent la période d'apprentissage et compliquent l'implication dans cette méthodologie.

La gestion des exigences est aujourd'hui reconnue de plus en plus importante. Elle représente la possibilité de définir des exigences, mais aussi de les représenter avec une forme compréhensible. Pour avoir un processus efficace, l'utilisation d'outils de modélisation mais aussi d'interprétations automatiques est indispensable. Ceux-ci offrent des possibilités de documentation, de gestion du changement et d'intégration d'autres outils en fonction de l'étape du projet dans lequel nous nous situons. Encore une fois, les méthodologies proposées ne sont pas toujours supportées par des outils, ce qui est primordial pour une implication importante des utilisateurs.

E. Approche MDA

1. Généralités

L'Object Management Group (OMG) a formalisé, en 2001, un ensemble de préconisations dans l'objectif de résoudre les problèmes de productivité, de portabilité et d'inter-opérabilités rencontrés au cours du développement de projets informatiques [KLE 03]. Ces préconisations doivent aussi faciliter le développement d'applications en mobilisant les connaissances métiers et/ou techniques existantes.

De ces réflexions est issue l'approche « Model Driven Architecture » (MDA) qui spécifie le cadre d'utilisation des modèles lors du développement d'un projet informatique [MIL 01]. En 2003, un guide de la démarche est édité par l'OMG [MIL 03] expliquant comment mettre en œuvre l'approche MDA et introduit quelques nouveaux concepts.

Afin de promouvoir l'approche MDA, l'OMG indique qu'un développement effectué suivant cette méthodologie permet de réaliser des gains de productivité et de qualité. Il est important de spécifier que MDA est une variante plus spécialisée que le MDE (« Model Driven Engineering ») ou IDM (« Ingénierie Dirigée par les Modèles ») en français. Cette approche est présentée plus longuement par la suite.

2. Description

L'approche MDA préconise la séparation des spécifications métiers propre à un domaine des spécifications techniques puis des spécifications propre à une technologie. Pour réaliser celle-ci, l'approche MDA définit deux types de modèles :

a. Les modèles indépendants des plates-formes (ou « Platform Independent Model ») : PIM. Il est indépendant de toute plate-forme et ne contient pas d'informations sur les technologies qui seront utilisées pour déployer l'application. À ce niveau, le formalisme utilisé pour exprimer un PIM est un diagramme de classes en UML qui peut-être couplé avec un langage de contrainte comme OCL (« Object Constraint Language »).

b. Les modèles spécifiques à une plate-forme (ou « Platform Specific Model ») : PSM. Le PSM sert essentiellement de base à la génération de code exécutable vers la ou les plates-formes techniques.

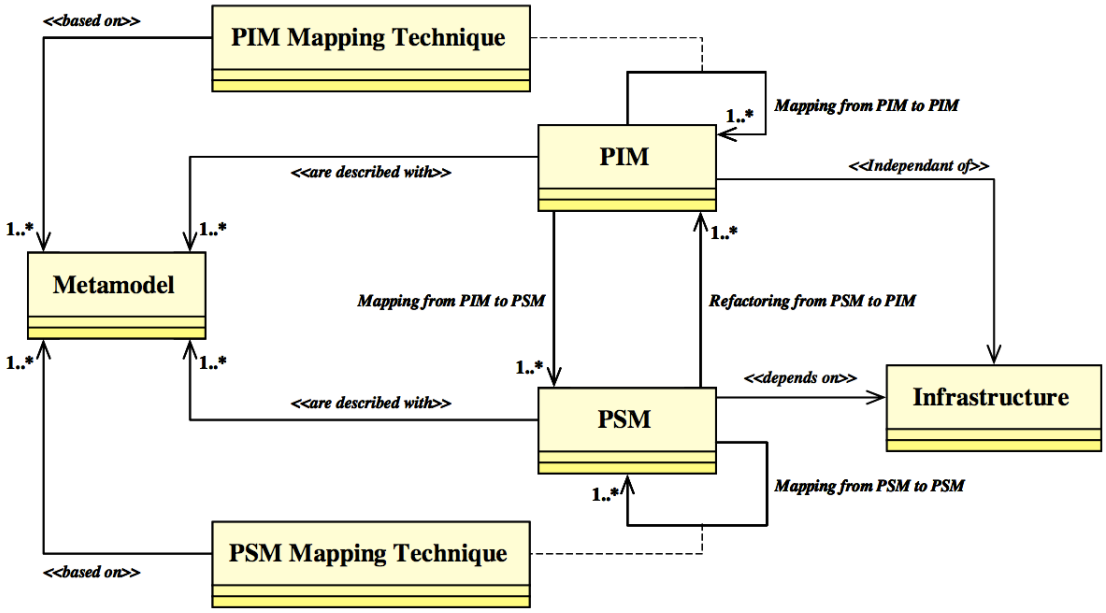


Figure 0-14 : Méta-modèle de l'approche MDA [MIR 06]

Ces modèles constituent l'architecture MDA (cf. Figure 0-14). Nous pouvons y voir que les modèles PIM et PSM peuvent être décrits par plusieurs méta-modèles en fonction du degré de raffinement de ces modèles.

En 2003, un nouveau type de modèle a été défini et ajouté à l'architecture MDA afin de combler le fossé existant entre les experts métier et les experts de la conception [MIL 03]. Ce nouveau modèle, nommé « Computation Independent Model » (CIM), a pour objectif d'être indépendant de la programmation et permet de décrire les exigences. C'est le modèle métier ou le modèle du domaine d'application. Le CIM permet la vision du système dans l'environnement où il opèrera, mais sans rentrer dans le détail de la structure du système, ni de son implémentation.

Une caractéristique importante du CIM est que le vocabulaire familier des acteurs du domaine est utilisé pour sa spécification [MIL 03]. Celui-ci doit servir d'aide à la compréhension du problème mais aussi comme référentiel en ce qui concerne le vocabulaire à utiliser dans d'autres modèles.

Pour terminer, un dernier type de modèle est proposé par l'OMG : le modèle de description de la plate-forme (« Platform Description Model ») ou PDM. Cette notion n'est pas encore bien définie par l'OMG, pour l'instant il s'agit plus d'une piste de recherche. Un PDM contient des informations pour la transformation de modèles vers une plate-forme en particulier et il est spécifique à celle-ci [GRA 07]. C'est un modèle de transformation qui va permettre le passage du PIM vers le PSM. Dans l'idéal, chaque fournisseur de plate-forme devrait le proposer [BEZ 03].

L'approche MDA introduit aussi la notion de transformation de modèles qui permet de faire évoluer les modèles au cours du développement. Celles-ci peuvent être manuelles ou automatiques. Cependant, les recherches actuelles sont principalement axées sur l'automatisation de celles-ci afin d'augmenter la productivité du développement. Deux types de transformations sont utilisables dans MDA :

a. Les transformations de raffinement : elles sont utilisées lorsque le modèle source et cible sont de même type (PIM/PIM et PSM/PSM). Celles-ci peuvent être appliquées par exemple, pour l'application de patrons de conception[3].
b. Les transformations de transposition : elles sont utilisées entre deux modèles de types différents. Nous pourrons définir des transformations d'application sur une plate-forme lorsque nous obtenons un PSM à partir d'un PIM, ou des transformations de refactorisation lorsque nous obtenons un PIM à partir d'un PSM.

Il est important de noter qu'il est possible d'obtenir plusieurs PSM à partir d'un même PIM en utilisant des transformations de transposition différentes. Cet

[3] Un patron de conception est un concept de génie logiciel destiné à résoudre les problèmes récurrents suivant le paradigme objet. Les patrons de conception décrivent des solutions standard pour répondre à des problèmes d'architecture et de conception des logiciels. Les patrons de conception les plus connus sont au nombre de 23. Ils sont couramment appelés « patrons GoF » (de « Gang of Four », d'après les quatre créateurs du concept [GAM 95]).

intérêt nous permettra de dériver une même spécification fonctionnelle sur plusieurs environnements technologiques.

3. Généralisation à l'ingénierie dirigée par les modèles

Séparer explicitement solution fonctionnelle et solution technologique est nécessaire pour faciliter le déploiement d'un logiciel sur différentes plateformes d'exécution. Des liens peuvent ensuite être tissés entre les modèles pour permettre de les assembler de manière automatisée et produire le logiciel final. Les avantages annoncés de l'IDM sont nombreux : indépendance vis à vis des évolutions technologiques, meilleure maîtrise de la complexité, meilleure réutilisation etc. [BEZ 04] De plus, l'IDM permet de se concentrer sur une préoccupation plus abstraite que la programmation classique. Il s'agit d'une forme d'ingénierie générative dans laquelle tout ou partie d'une application est engendrée à partir de modèles.

Un modèle est une abstraction suffisante pour comprendre le système modélisé et répondre aux questions que nous nous posons sur lui. Un système peut être décrit par différents modèles liés les uns aux autres. L'idée phare est d'utiliser autant de langages de modélisation différents (DSL) afin de définir le système étudié. La définition de ces DSL, appelée méta-modélisation, est donc une problématique clé de cette nouvelle ingénierie. Par ailleurs, afin de rendre opérationnels les modèles (pour la génération de code, de documentation et de test, la validation, la vérification, l'exécution, etc.), une autre problématique clé est celle de la transformation de modèles.

La notion de modèle dans l'IDM fait explicitement référence à la notion de langage. En effet, pour qu'un modèle soit productif, il doit pouvoir être manipulé par une machine. Le langage dans lequel ce modèle est exprimé doit donc être clairement défini. De manière naturelle, la définition d'un langage de modélisation a pris la forme d'un modèle, appelé méta-modèle. Afin d'illustrer ce propos, il est possible de faire une analogie dans le domaine de la cartographie. Il est indispensable d'associer à chaque carte la description du « langage » utilisé. Cette description est usuellement appelée une légende. La carte doit, pour être utilisable, être conforme à cette légende. Plusieurs cartes peuvent être

conformes à une même légende. La légende est alors considérée comme un modèle représentant cet ensemble de cartes et à laquelle chacune d'entre elles doit se conformer.

De nombreux méta-modèles ont émergés afin d'apporter chacun leurs spécificités dans un domaine particulier. Devant le danger de voir émerger indépendamment et de manière incompatible cette grande variété de méta-modèles, il y avait un besoin urgent de donner un cadre général pour leur description. La réponse fut donc d'offrir un langage de définition de méta-modèles qui prit lui-même la forme d'un modèle : ce fut le méta-méta-modèle MOF (Meta-Object Facility) [OMG 06]. En tant que modèle, il doit également être défini à partir d'un langage de modélisation. Pour limiter le nombre de niveaux d'abstraction, celui-ci a la capacité de se décrire lui-même.

C'est sur ces principes que se base l'organisation de la modélisation de l'OMG généralement décrite sous une forme pyramidale (cf. Figure 0-15). Le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les méta-modèles permettant la définition de ces modèles (par exemple UML) constituent le niveau M2. Enfin, le méta-méta-modèle, unique, est représenté au sommet de la pyramide (niveau M3).

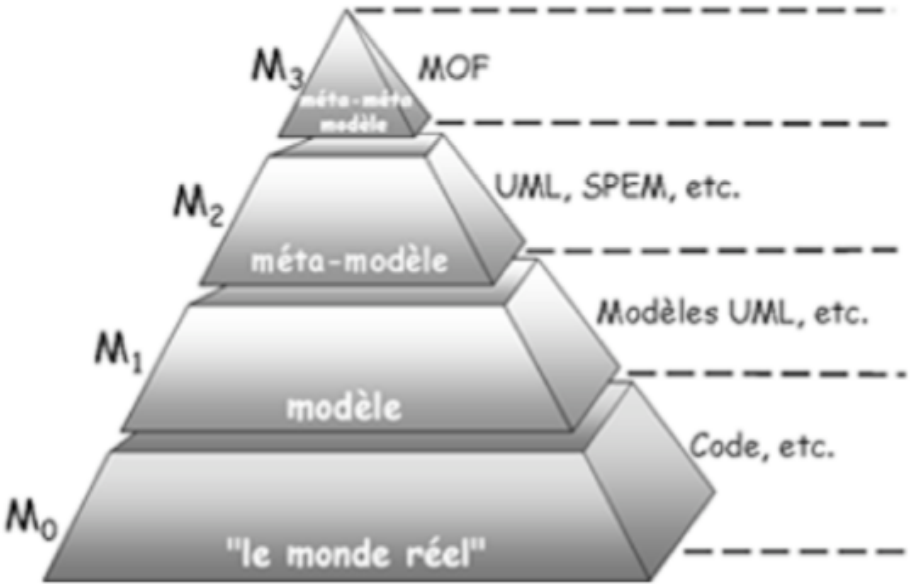


Figure 0-15 : Pyramide de modélisation de l'OMG

Un point important est de séparer clairement les approches IDM du formalisme UML, et de l'utilisation qui en est faite dans le MDA. En effet, non seulement la portée de l'IDM est plus large que celle d'UML mais la vision de l'IDM est aussi très différente de celle d'UML, parfois même en contradiction. UML est un standard assez monolithique obtenu par consensus a maxima, dont nous devons réduire ou étendre la portée à l'aide de mécanismes comme les profils. Ces mécanismes n'ont pas tous la précision souhaitable et mènent parfois à des contorsions dangereuses pour « rester »dans le monde UML. Au contraire, l'IDM favorise la définition de langages de modélisation dédiés à un domaine particulier (DSL) offrant ainsi aux utilisateurs des concepts propres à leur métier et dont ils ont la maîtrise. Ces langages sont généralement de petite taille et doivent être facilement manipulables, transformables, combinables, etc.

4. Panorama des solutions de transformation de modèles

Une des problématiques de l'IDM est de rendre opérationnel les modèles à l'aide de transformations [COM 08]. D'autre part, l'approche MDA repose sur le principe de la création d'un PIM pouvant être raffiné en un ou plusieurs PSM. Les méthodes de transformation sont là aussi indispensables pour changer de niveau d'abstraction (transformation verticale), dans le cas du passage de PIM à PSM et inversement. Il est également possible de rester au même niveau d'abstraction

(transformation horizontale) dans le cas de transformation PIM à PIM ou PSM à PSM [GER 02].

De nombreux langages sont à ce jour disponibles pour écrire des transformations de modèles. Nous retrouvons d'abord les langages généralistes qui s'appuie directement sur la représentation abstraite du modèle, telle que l'API Java[4] d'EMF.

Afin d'abstraire la définition de transformations de modèles, l'idée a été de définir des langages dédiés à la transformation de modèles. Nous pouvons citer le langage ATL (« ATLAS Transformation Language ») [JOU 05] dont nous reparlerons par la suite. Il s'agit d'un langage hybride (déclaratif et impératif) qui permet de définir une transformation de modèles à modèle sous la forme d'un ensemble de règles. D'autres langages de transformation déclaratifs sont proposés tels que Kermeta Transformation Rule (KTR).

Percevant leur importance stratégique, l'OMG a cherché à « unifier » les langages de transformation. L'objectif de l'OMG est, comme pour le langage UML, de définir un langage de transformation de modèles adopté par la majorité de la communauté informatique. L'OMG a défini le standard QVT (*Query/View/Transformation*) [OMG 02]. Le méta-modèle de QVT fait apparaître trois sous langages pour la transformation de modèles, caractérisés par le paradigme mis en œuvre pour la définition des transformations (déclaratif, impératif et hybride).

Il existe une catégorie bien particulière de transformation de modèles. Elle fait référence aux technologies permettant de générer du code : les transformations « model to text ». Plusieurs propositions font appel à l'utilisation de *templates*. Elles consistent à appliquer des masques au modèle utilisé et de générer un code spécifique pour chaque élément du modèle. Nous pouvons faire référence à

---

[4] Le langage Java est un langage de programmation informatique orienté objet ayant la particularité de simplifier le portage des logiciels l'utilisant sur plusieurs systèmes d'exploitation.

Acceleo[5], *Xpand*[6], etc. Il est également possible d'utiliser une approche plus programmatique pour la navigation dans le modèle source. Ces approches s'inspirent généralement de langage existant. Nous pouvons citer la technologie *JET*[7] s'inspirant fortement de la syntaxe *JSP*[8].

5. Comparaison d'un processus de développement traditionnel et d'un processus de développement appliquant la méthode MDA

Afin de confirmer ou démentir les allégations de l'OMG concernant le gain de productivité attendu avec l'approche MDA, la société américaine *Compuware Corporation* a demandé au cabinet de conseil *The Middleware Company* de faire une étude comparative pour vérifier ces affirmations [MID 03].

Pour ce faire, le cabinet de conseil a demandé à deux équipes de niveau de compétences équivalent de développer une application de commerce électronique d'animaux domestiques *PetStore* : la première en mettant en œuvre un processus de développement traditionnel et la seconde en utilisant l'approche MDA. Le cahier des charges transmis aux deux équipes était le même. Le Tableau 0-2 : Comparaison du processus de développement traditionnel et approche MDA montre les durées planifiées et réalisées par ces équipes.

		Durée	
		Planifiée	*Réalisée*
Équipe	*Traditionnelle*	499h	507h
	MDA	442h **(-11%)**	330h **(-35%)**

Tableau 0-2 : Comparaison du processus de développement traditionnel et approche MDA

Les résultats parlent d'eux mêmes. Initialement évalué à 11%, le gain de productivité de l'équipe MDA a été triplé. Selon l'un des programmeurs de l'équipe MDA, ce gain aurait pu être plus élevé car le temps de la réalisation

---

[5] http://www.acceleo.org/
[6] http://wiki.eclipse.org/Xpand
[7] JET (Java Emitter Templates) : http://www.eclipse.org/emft/projects/jet/
[8] JSP (Java Server Pages) est une technique basée sur Java qui permet aux développeurs de générer dynamiquement du code HTML, XML ou tout autre type de page web. Source : http://fr.wikipedia.org/wiki/JavaServer_Pages

aurait pu être réduit de 10 à 20 %. En effet, au démarrage du projet l'équipe MDA n'avait aucune expérience de cette approche et a dû, dans un premier temps, se former aux principes et acquérir la maîtrise des outils MDA.

Un autre résultat ressort de cette étude : l'équipe traditionnelle, tout au long du développement, a été amenée à corriger des bogues alors que l'équipe MDA n'a pas eu ce problème. Les analystes du cabinet conseil encadrant l'étude attribuent cette absence de bogue à l'utilisation intensive de la génération automatique de code.

6. Synthèse

Le principe clé de la méthodologie MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement d'une application. L'objectif majeur de MDA est l'élaboration de modèles pérennes, indépendants des détails techniques des plates-formes d'exécution, afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité. [BLA 05]

La démarche MDA suscite un réel intérêt chez bon nombre d'industriels et de développeurs. En effet, cette démarche est prometteuse et répond à des attentes légitimes non comblées par les technologies orientées objet ou composant. Elle autorise la séparation de la logique métier de l'organisation, de son implémentation physique. Cette nouvelle approche bouleverse complètement notre façon de concevoir une application et ouvre de nouveaux horizons pour le génie logiciel qui ne sera plus dépendant des évolutions technologiques.

Pour que le MDA se diffuse auprès des développeurs, le savoir doit être essaimé et de nouveaux outils doivent être développés. Pour produire ces outils qui font encore aujourd'hui défaut, plusieurs projets ont été lancés pour que l'approche MDA tienne ses promesses. Ces outils permettront l'automatisation des transformations ainsi que la génération automatique de code à partir de modèles, permettant aux architectes de se consacrer pleinement aux tâches de modélisation métier. La démarche MDA deviendra une architecture viable, dès lors que ces transformations seront concrétisées et intégrées dans les outils.

Un point important est de séparer clairement les approches IDM du formalisme UML, et de l'utilisation qui en est faite dans le MDA. En effet, non seulement la portée de l'IDM est plus large que celle d'UML mais la vision de l'IDM est aussi très différente de celle d'UML, parfois même en contradiction. UML est un standard assez monolithique obtenu par consensus, dont nous devons réduire ou étendre la portée à l'aide de mécanismes comme les profils [OMG 07]. Ces mécanismes n'ont pas tous la précision souhaitable et mènent parfois à des contorsions dangereuses pour « rester » dans le monde UML. Au contraire, l'IDM favorise la définition de langages de modélisation dédiés à un domaine particulier (DSL) offrant ainsi aux utilisateurs des concepts propres à leur métier et dont ils ont la maîtrise. Ces langages sont généralement de petite taille et doivent être facilement manipulables, transformables, combinables, etc.

F. Conclusion

Dans ce chapitre nous avons présenté les principaux cycles de conception de SI. Nous avons tenté de situer les modalités recommandées de l'intégration des utilisateurs dans ces cycles. Une des principales conclusions de cette analyse est l'apport reconnu des processus itératifs pour une conception répondant aux exigences des utilisateurs. En effet, l'intégration de phases de prototypage et de validation successives favorise l'implication des utilisateurs au cours de la conception.

Afin de favoriser cette intégration, il est possible de s'appuyer sur la norme de conception centrée utilisateur ISO 13407. Sans être un modèle de conception, cette norme propose des recommandations pour une conception répondant au mieux aux exigences des utilisateurs finaux du système à concevoir. Il paraît relativement difficile d'appliquer cette démarche à des situations réelles de conception. En effet, les fortes contraintes pesant sur un projet de développement, les pratiques de conception habituelles et les possibilités technologiques peuvent rendre difficile l'application stricte d'une démarche centrée sur l'utilisateur. Il s'agit alors de garder les lignes directrices de cette démarche, c'est-à-dire le cycle itératif de conception et la validation par un retour d'information de la part des utilisateurs à chaque itération.

Afin d'évaluer l'utilisabilité d'un système, différentes méthodes sont proposées. Les produits développés doivent permettra aux utilisateurs d'être efficace et efficient. Des analyses de l'usage ou la mise en place de méthodes créatives peuvent devenir des sources d'information fiables et très riches. Malheureusement, il est peu concevable de réaliser de lourdes investigations à chaque itération du processus de conception. Il faut donc proposer des solutions plus flexibles et moins coûteuses.

L'expression des besoins utilisateur est également un levier important pour améliorer l'utilisabilité d'un SII. Dans ce domaine, l'ingénierie des besoins propose des approches relativement complexes et difficiles à mettre en œuvre rapidement au sein d'une organisation. De plus, il est important de favoriser la diffusion des besoins entre les acteurs mais également à chaque étape de la conception. Il est donc primordial de disposer d'outils d'interprétations automatiques. Ceux-ci offrent des possibilités de documentation, de gestion du changement et d'intégration d'autres outils en fonction de l'étape du projet dans lequel nous nous situons.

L'approche MDA cherche à concevoir des SI de meilleure qualité mais en se concentrant sur la qualité du code et le gain de productivité potentiel apportés par les différents mécanismes automatiques. Bien que MDA préconise d'utiliser un modèle indépendant de la technologie (CIM), celui-ci est peu exploité et ne rentre pas ou peu dans les mécanismes de transformation automatiques traditionnels.

Dans la suite, nous proposerons une approche dirigée par les modèles pour la spécification et la validation des besoins. En raison de la complexité des langages de spécification des besoins orientés par les buts, nous proposerons un langage simplifié. Nous présenterons ensuite un mécanisme générique et automatique permettant d'interpréter une définition du besoin afin d'améliorer la diffusion et le retour d'information par et vers les utilisateurs.

Le socle de cette démarche repose sur un langage commun de spécification des besoins. Notre hypothèse est qu'un langage compréhensible par les utilisateurs finaux et les concepteurs est un pas vers une meilleure diffusion et

compréhension des besoins. Ce langage sera supporté par un outil et un ensemble de mécanismes d'interprétations pour adapter la visualisation des besoins au profil qui consulte la spécification afin de favoriser le retour d'informations.

SUN : Sustainable User Need

A. Introduction

Nous proposons la méthode SUN ; une approche méthodologique visant à soutenir le recueil des besoins utilisateurs, leur validation ainsi que leur diffusion dans le contexte d'un projet de conception d'un SII. La propagation de ces besoins, entre les différents intervenants lors des différentes phases d'un projet de développement, repose sur un langage d'expression des besoins compréhensible par tous et d'un mécanisme d'interprétation permettant d'adapter la visualisation au profil qui l'utilise et à l'étape concernée.

Dans le premier chapitre de ce mémoire, nous avons identifié l'insuffisance de méthodologie et de langage commun entre les différents acteurs d'un projet de conception. Suite à ce constat, nous avons identifié deux axes de connaissances essentiels à la définition d'un projet :

a. Connaissances relatives au domaine d'application du système ;
b. Connaissances relatives aux activités utilisateurs réalisées à travers le système.

Ces deux axes sont évidemment indissociables, il faut donc proposer un troisième axe transverse permettant de faire le lien entre les deux précédents. Cet axe est relatif aux politiques d'accès permettant de connaître les concepts métier manipulés à chacune des activités utilisateur.

Afin d'améliorer au mieux la diffusion des besoins fonctionnels, il est utile d'adapter la visualisation au profil qui le consulte mais aussi à l'étape de conception courante. En proposant des moyens de présentation familiers aux intervenants, nous ne pouvons que favoriser leur intégration et leur compréhension. Toutefois, cette adaptation ne doit pas être une activité consommatrice dans le processus de conception. Pour cela, nous préconisons d'utiliser l'approche IDM permettant d'automatiser l'ensemble de ces mécanismes.

L'hypothèse avancée est que le choix de formalismes de définition des besoins se répercute sur l'expression des connaissances et donc sur leur diffusion tout au long du processus de conception. Cette démarche repose sur un langage simplifié favorisant la compréhension et la définition des besoins. De plus, afin de diminuer la barrière créée par la proposition d'un nouveau langage, cette approche propose de transposer automatiquement les besoins définis dans le langage proposé dans un autre moyen de représentation adapté à chaque profil d'intervenant.

B. Les composantes de la méthodologie SUN

1. Le langage

a) Présentation

L'objectif principal dans la définition de ce langage est de trouver le bon rapport entre l'exhaustivité de ce langage et sa difficulté de compréhension. Nous faisons l'hypothèse qu'un nombre limité de concepts manipulables par l'utilisateur permettra une prise en main plus rapide et une meilleure compréhension du langage.

Le premier axe, à couvrir par notre proposition de langage, est relatif au domaine d'application du système. Nous cherchons à définir le modèle de données à un niveau fonctionnel du système. Pour définir celui-ci, nous avons à disposition deux grandes techniques :

a. Langages entité-association (E-A);
b. Langages orientés objet.

Les études actuelles montrent que les langages entité-association montrent des performances supérieures, que ce soit pendant l'activité de modélisation ou de lecture [TOP 02] [AGU 08]. C'est à Chen [CHE 76] que nous devons la définition des modèles E-A en s'appuyant sur deux concepts : entité et relation. Ces modèles tentent de séparer la structure de données des traitements qu'elles peuvent subir [BAC 02]. De cette base, nous n'avons gardé que quelques

concepts mais ignoré ceux considérés comme trop complexe comme les associations ternaires ou les type-entités.

Définition 'Entité' :

Une entité est un objet, une chose concrète ou abstraite qui peut être reconnue distinctement.

Définition 'Association' :

Une association (ou une relation) est un lien entre plusieurs entités.

Définition 'Attribut' :

Un attribut (ou une propriété) est une caractéristique associée à une entité. Celui-ci peut être de nature textuelle, temporelle, numérique ou de nature autre.

Pour le second axe basé sur la définition des activités utilisateurs, nous nous sommes intéressés à l'ingénierie des besoins. Comme présenté dans le chapitre précédent, les langages proposés sont beaucoup plus simples à utiliser lorsqu'ils sont orientés par les objectifs plutôt que basés sur les scénarios [MIS 05]. De ces langages, nous avons seulement retenu le concept d'objectif et d'agent. En ce qui concerne les relations entre ces concepts, la notion de responsabilité entre un acteur et un objectif a été conservée ainsi que la notion de décomposition d'objectifs.

Définition 'Objectif' :

Un objectif représente ce que nous cherche à atteindre. Selon [JAC 95], un but est une déclaration 'optative' qui exprime ce que l'on veut, un état ou un résultat que l'on cherche à atteindre. Un objectif peut être sous la responsabilité d'un ou plusieurs agents.

Définition 'Agent' :

Un agent est un composant ayant la capacité d'agir par lui-même et de satisfaire ses propres objectifs. Celui-ci peut être humain ou non. Dans le cas d'un agent humain, l'objet défini fera référence à une catégorie d'utilisateur dans le système à modéliser.

À ces concepts, nous avons rajouté un type d'association retrouvé relativement rarement dans les méthodes existantes. Il s'agit du concept de précédence qui permet de décrire des contraintes d'antériorité entre la réalisation de deux objectifs. Ce type d'association nous permet de décrire simplement des processus métier.

Le troisième axe permet de lier la modélisation des activités réalisées à la modélisation du domaine métier qui s'inspire du modèle E-A à l'aide d'objectif. Cet axe repose sur le concept de privilège. Pour chacune des activités représentées à l'aide d'un objectif, une politique d'accès lui est adjointe. Celle-ci est explicitée à l'aide de privilèges. Nous proposons ces deux définitions :

Définition 'Privilège' :

Un privilège est un droit accordé sur une entité ou sur un attribut. Les droits de lecture et de mise à jour peuvent être accordés à une entité ou à un attribut. Tandis que les droits de création et de suppression sont spécifiques à une entité.

Définition 'Politique d'accès' :

Une politique d'accès est un ensemble de privilèges permettant de définir les informations potentiellement consommées et produites par le ou les responsables d'un objectif.

Comme nous l'avons vu dans la section précédente, l'IDM préconise l'utilisation de petits langages de modélisation, dédiés chacun à un domaine particulier. La méta-modélisation, activité correspondant à définir un DSL, doit donc être étudiée et maîtrisée. Dans le contexte de l'informatique, nous distinguons généralement la syntaxe concrète (CS), manipulée par l'utilisateur du langage, de la syntaxe abstraite (AS) qui est la représentation interne (d'un programme ou d'un modèle) manipulée par l'ordinateur [ASU 86].

b) Syntaxe abstraite

Dans le contexte de l'IDM, la syntaxe abstraite est placée au cœur de la description d'un langage de modélisation. Elle est généralement décrite en premier et sert de base pour définir la syntaxe concrète.

La syntaxe abstraite (AS) d'un langage de modélisation exprime, de manière structurelle, l'ensemble de ses concepts et leurs relations. Les langages de méta-modélisation tels que le standard MOF de l'OMG [OMG 06], offrent les concepts et les relations élémentaires qui permettent de décrire un méta-modèle représentant la syntaxe abstraite d'un langage de modélisation. Pour définir cette syntaxe, nous disposons à ce jour de nombreux environnements et langages de méta-modélisation : Eclipse-EMF/Ecore [BUD 03], GME/MetaGME [LED 01], AMMA/KM3 [JOU 06a][ATL 05], XMF-Mosaic/Xcore [CLA 04] ou Kermeta [MUL 05a]. Tous ces langages reposent toutefois sur les mêmes constructions élémentaires (cf. Figure 0-1). S'inspirant de l'approche orientée objet, les langages de méta-modélisation objet offre le concept de classe (*Class*) pour définir les concepts d'un DSL (cf. Figure 0-1). Une classe est composée de propriétés (*Property*) qui la caractérisent. Une propriété est appelée *référence* lorsqu'elle est typée (*TypedElement*) par une autre classe, et *attribut* lorsqu'elle est typée par un type de donnée (par exemple : booléen, chaîne de caractères et entier).

Pour définir la syntaxe abstraite de notre méta-modèle définissant notre langage, nous avons utilisé l'éditeur graphique du projet *Topcased*[9] permettant la description de méta-modèles à l'aide du langage de méta-modélisation eCore (cf. Figure 0-2). Nous retrouvons dans le méta-modèle proposé les trois axes cités ci-dessus.

[9] Topcased est un projet atelier logiciel open source intégré à l'IDE Eclipse visant à fournir un ensemble d'outils de génie logiciel pour le développement de systèmes critiques temps réels. Son but est de proposer une solution pérenne à faible coût en s'inscrivant dans le processus. http://topcased.gforge.enseeiht.fr/

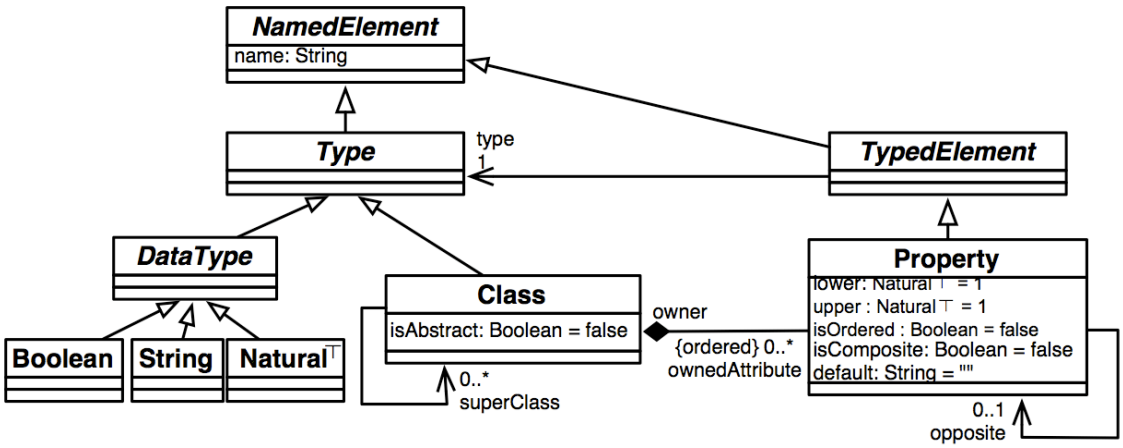


Figure 0-1 : Concepts principaux de méta-modélisation (EMOF 2.0)

La définition du domaine (voir la partie jaune sur la Figure 0-2) peut se faire à l'aide des entités (*Entity*) caractérisées par des attributs (*Attribute*) et reliées potentiellement entre elles par des associations (*RelationShip*). Une activité peut être représentée par un objectif (*Goal*) décomposable et sous la responsabilité d'un acteur (*Agent*). Chaque activité peut intervenir dans un ou plusieurs processus (*Process*) représentant ainsi une étape (*GoalStep*) de celui-ci. Pour lier les activités et le domaine du SII à modéliser, il est possible de créer des polices d'accès (*PrivilegeGroup*) composées d'un ensemble de privilèges (*Privilege*).

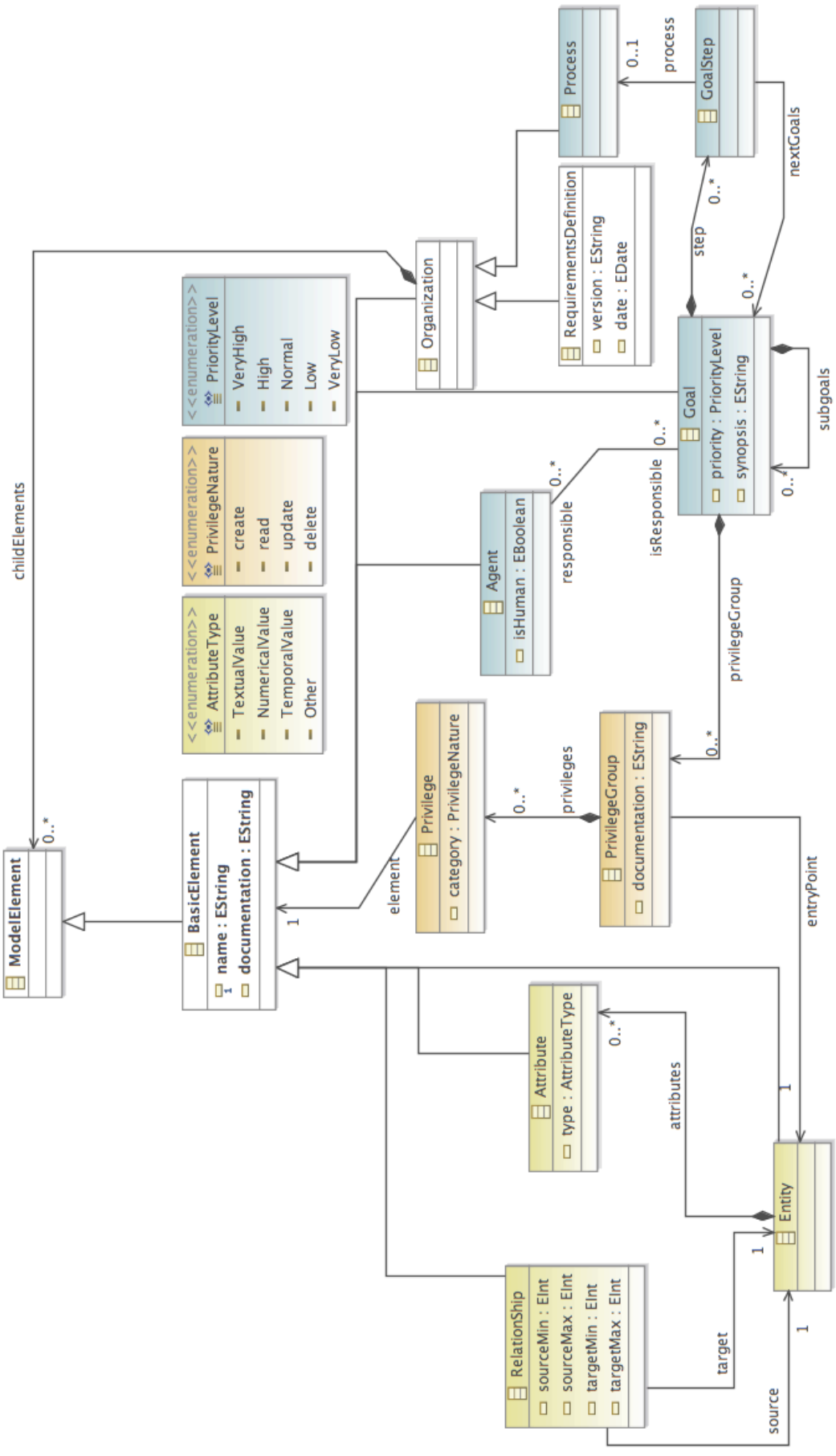

ModelElement
childElements
0..*
BasicElement
name : EString
documentation : EString
1
element
Privilege
category : PrivilegeNature
0..*
privileges
PrivilegeGroup
documentation : EString
0..*
privilegeGroup
Agent
isHuman : EBoolean
responsible
0..*
isResponsible
0..*
Goal
priority : PriorityLevel
synopsis : EString
0..*
subgoals
step
0..*
nextGoals
Process
0..1
process
GoalStep
RequirementsDefinition
version : EString
date : EDate
Organization
<<enumeration>>
PriorityLevel
VeryHigh
High
Normal
Low
VeryLow
<<enumeration>>
PrivilegeNature
create
read
update
delete
<<enumeration>>
AttributeType
TextualValue
NumericalValue
TemporalValue
Other
Attribute
type : AttributeType
0..*
attributes
entryPoint
1
Entity
RelationShip
sourceMin : EInt
sourceMax : EInt
targetMin : EInt
targetMax : EInt
target
source
1
1

Figure 0-2 : Méta-modèle SUN au format eCore

c) Syntaxe concrète

Les syntaxes concrètes (CS) d'un langage fournissent à l'utilisateur un ou plusieurs formalismes, graphiques et/ou textuels, pour manipuler les concepts de la syntaxe abstraite et ainsi en créer des « instances ». Le modèle ainsi obtenu sera conforme à la structure définie par la syntaxe abstraite.

La spécification d'une syntaxe concrète est à ce jour bien maîtrisée et outillée. Il existe en effet de nombreux projets qui s'y consacrent, principalement basés sur EMF [BUD 03]: GMF[10] [MOO 04], TOPCASED[11] [FAR 06], Merlin Generator[12], GEMS[13], TIGER[14] [EHR 04], etc. Si ces derniers sont principalement graphiques, des projets récents permettent de définir des modèles de syntaxe concrète textuelle. Nous citons par exemple les travaux des équipes INRIA Triskell (Sintaks[15] [MUL 05b]) et ATLAS (TCS[16] [JOU 06b]) qui proposent des générateurs automatiques d'éditeurs pour des syntaxes concrètes textuelles.

Le premier outil proposé a été entièrement développé, mais à cause de certaines contraintes, nous avons fait le choix de définir un nouvel outil de modélisation. Afin d'être conforme à notre approche, nous avons fait le choix d'utiliser des méthodologies déjà existantes dans l'IDM telle que TOPCASED.

---

[10] http://www.eclipse.org/modeling/gmp/
[11] http://www.topcased.org/
[12] http://merlingenerator.sourceforge.net/
[13] http://www.eclipse.org/gmt/gems/
[14] http://user.cs.tu-berlin.de/~tigerprj/
[15] http://www.kermeta.org/sintaks
[16] http://www.eclipse.org/gmt/tcs/

Outil de modélisation web

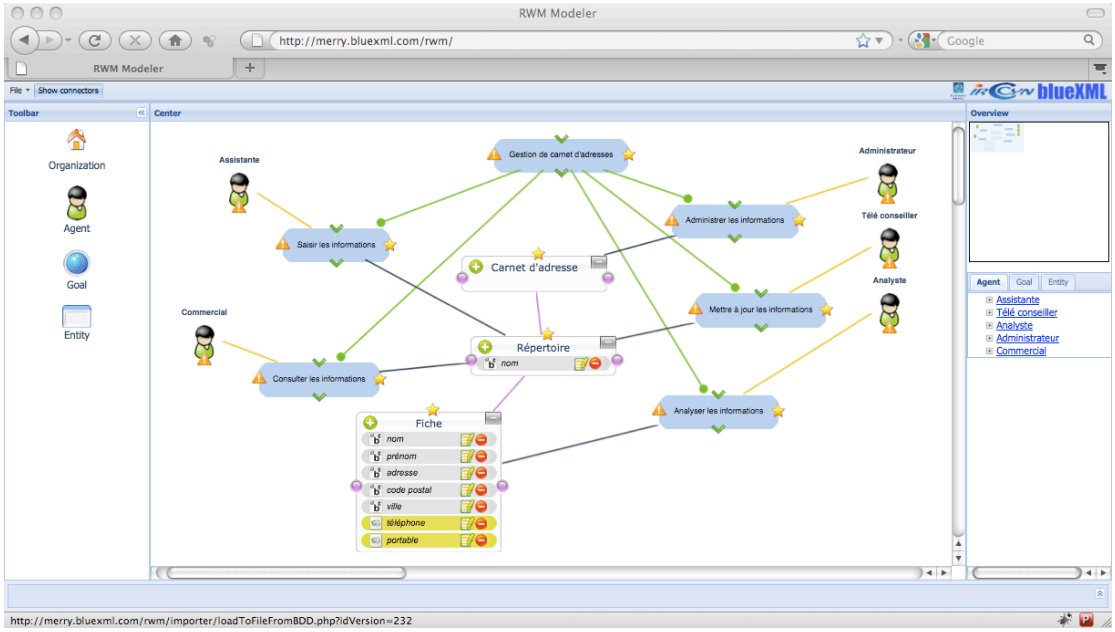


Figure 0-3 : Outil de modélisation web

Le premier outil de modélisation permettant d'utiliser le langage proposé (dont la syntaxe abstraite est définie sur la Figure 0-2) était un modeleur Web, c'est-à-dire directement utilisable dans un navigateur. Cette possibilité permet de faciliter l'expression des besoins par des experts métier en simplifiant son accès. Il n'existe aujourd'hui aucun outil ou aucune méthodologie pour développer un modeleur Web. Nous avons donc du développer cet outil entièrement et sans aucun assistant de génération en utilisant la librairie *Open-jACOB Draw2D*[17]. Le temps nécessaire à sa réalisation et à sa maintenance a été très important en raison de l'immaturité des technologies utilisées dans le contexte de la création d'un modeleur. De plus, la compréhension des modèles avec cette première version était très difficile en raison de l'absence de fonctionnalités indispensables, telles que la recherche. Nous avons donc fait le choix de développer un nouvel outil de modélisation complémentaire de celui-ci mais présentant beaucoup de facilités en terme d'évolution et d'intégration en utilisant les outils cités ci-dessus.

[17] http://www.draw2d.org/

Outil de modélisation Eclipse

TOPCASED présente les caractéristiques adéquates à notre besoin. Nous cherchons un outil nous permettant de générer rapidement un éditeur graphique à partir de notre langage modélisé à l'aide du méta-modèle Ecore. De plus, les résultats de ce projet ont déjà été testés et expérimentés industriellement en montrant de très bons résultats. En effet, initialement destiné à la création d'environnement pour l'industrie aéronautique, celui-ci s'est illustré en proposant des éditeurs pour le génie logiciel ou l'ingénierie système avec l'éditeur SysML. Cet outil permet, pour un modèle Ecore donné, de définir une syntaxe concrète graphique et l'éditeur associé. La syntaxe concrète est décrite dans un modèle de configuration qui offre une grande liberté de personnalisation des éléments graphiques souhaités pour représenter les concepts. Le projet TOPCASED a été utilisé pour générer les éditeurs pour les langages Ecore, UML2, AADL, SAM, SYSML…

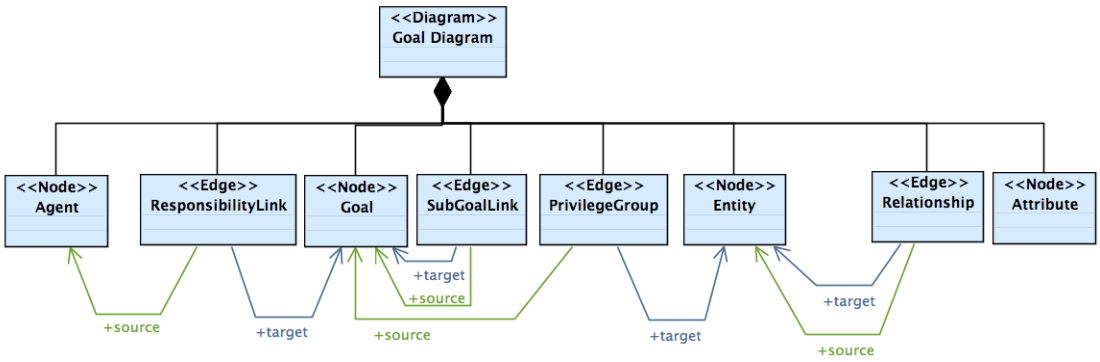


Figure 0-4 : Modèle de configuration de la syntaxe concrète

Pour notre langage, nous avons tout d'abord défini le modèle de notre syntaxe concrète (cf. la version simplifiée de ce modèle sur la Figure 0-4). Nous avons défini quatre éléments de type *Node* (ou boîte) : *Goal*, *Agent*, *Entity*, *Attribute*. Nous avons également défini un diagramme (*Goal Diagram*) qui correspond à un paquetage qui contiendra les autres éléments. Différents liens sont également créer entre les boîtes (*Edge*) : *ResponsibilityLink*, *SubGoalLink*, *PrivilegeGroup* et *Relationship*. Définir la syntaxe concrète d'un langage peut nécessiter l'emploi d'éléments additionnels ne correspondant à aucun concept abstrait. Par exemple,

ici il est indispensable d'ajouter *ResponsibilityLink* comme *Edge* pour relier un *Agent* à un *Goal*. *ResponsibilityLink* ne correspond pourtant à aucun concept de notre langage. Il s'agit de « colle » syntaxique dont la présence est obligatoire pour décrire un élément graphique. Il faut noter que les concepts de la syntaxe abstraite et ceux de la syntaxe concrète sont des concepts différents qui doivent être mis en correspondance. Nous avons employé les mêmes noms quand la correspondance était évidente. L'outil de modélisation engendré est présenté sur la Figure 0-5. Celui-ci a été personnalisé pour ressembler à ce qu'il est aujourd'hui.

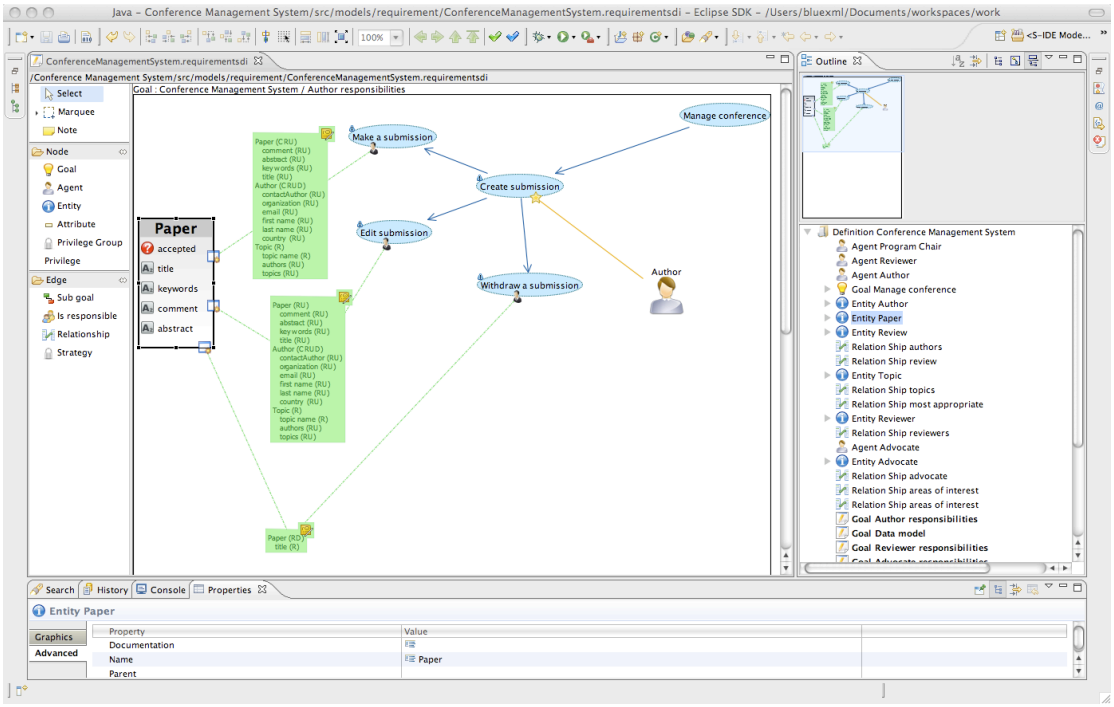


Figure 0-5 : Outil de modélisation basé sur Eclipse

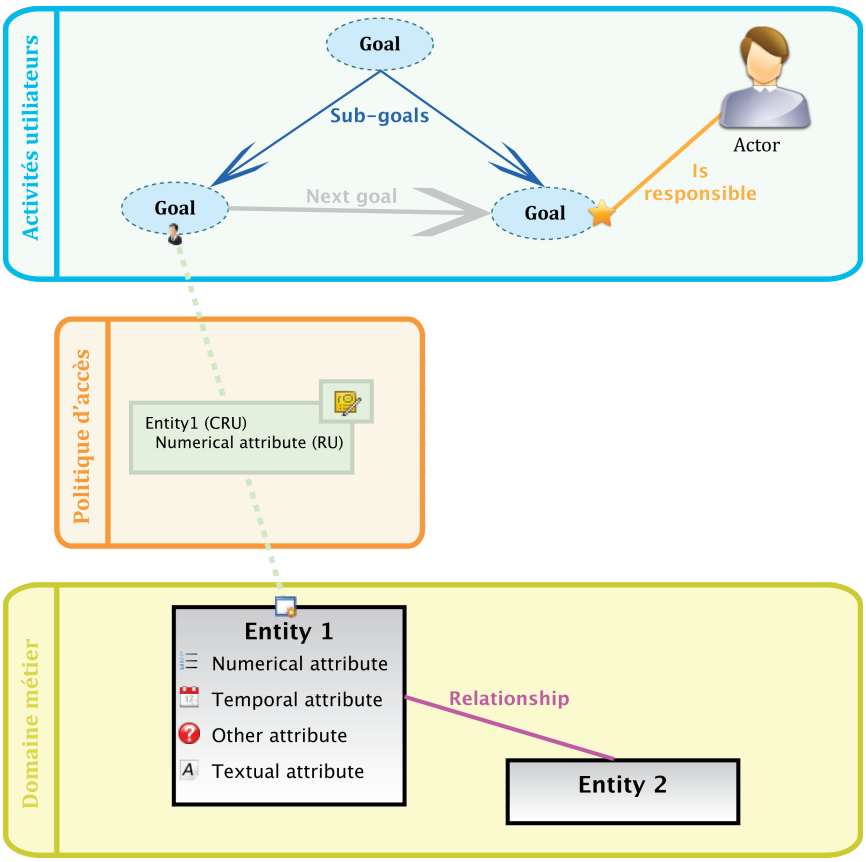


Figure 0-6 : Récapitulatif des éléments graphiques disponibles

Nous avons synthétisé l'ensemble des éléments graphiques dans la Figure 0-6. Nous pouvons y voir comment est représenté la modélisation d'activité à l'aide d'objectif, d'agent et des différentes natures de liens entre ces deux concepts. La notion d'entité et de relation est utile à la modélisation du domaine métier. Afin de relier ces deux axes, nous avons à disposition le concept de politique d'accès permettant de lier les objectifs au domaine métier.

A cet outil de modélisation, différentes fenêtres de dialogue ont été ajoutées telle que la fenêtre de dialogue permettant de décrire plus simplement des politiques d'accès (cf. Figure 0-7). À travers celle-ci, il est ainsi plus facile de naviguer dans la structure définie à l'aide des entités, attributs et associations.

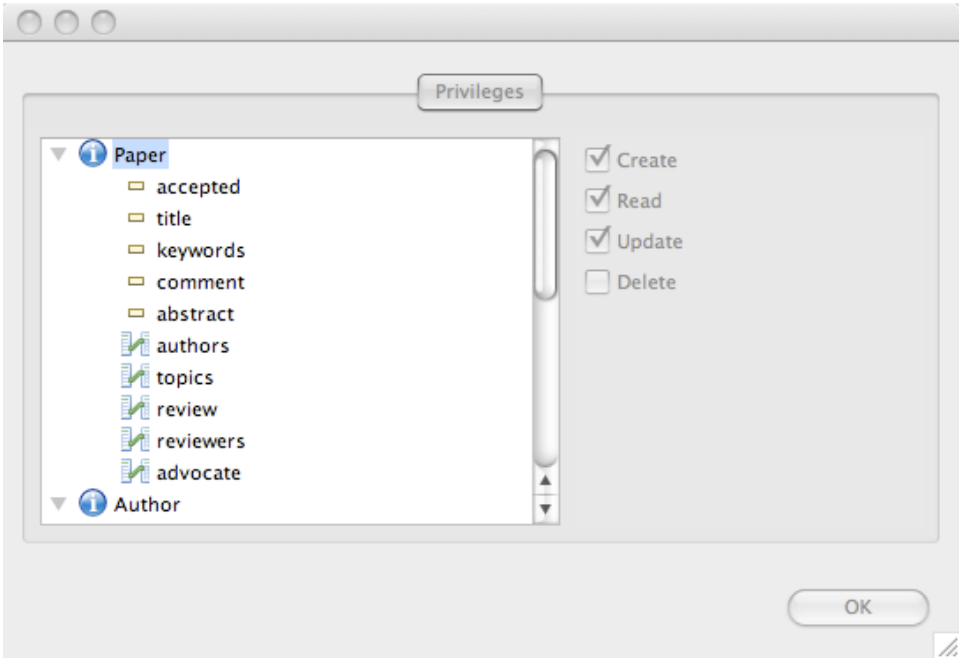


Figure 0-7 : Fenêtre d'édition des politiques d'accès

2. Le mécanisme d'interprétation

Afin de diminuer la résistance au changement face à cette proposition de langage, nous définissons le concept d'interprétation. Celui-ci nous permet d'adapter la visualisation de la spécification des besoins à chacun des profils, et cela en fonction de l'étape courante du processus de développement. Concrètement, l'apport d'un nouveau langage ou d'une nouvelle méthodologie entraine, généralement, une phase d'apprentissage nécessaire qui peut devenir très consommatrice en temps malgré l'objectif de simplicité du langage. Pour réduire cette phase, nous proposons de former seulement les gens en interaction directe avec le langage. Pour les autres profils, nous mettons en place un mécanisme d'interprétation qui permettra ainsi d'utiliser un moyen de visualisation connu par le profil concerné.

Ce mécanisme d'interprétation est générique et doit être adapté au projet sur lequel cette méthodologie est appliquée mais aussi aux participants impliqués et surtout aux informations que nous souhaitons partager. Toutefois, celui-ci doit présenter certaines caractéristiques :

a. Automatique : l'interprétation doit être entièrement ou quasi entièrement automatisée afin d'économiser le plus de temps possible. Une

interprétation devient intéressante à partir du moment où elle est utilisée de manière récurrente, elle doit donc être automatique.

b. Complète : le résultat obtenu à l'issu de l'interprétation doit être utilisable directement où à l'aide d'un outil.

Il est important de mesurer l'intérêt d'une interprétation avant sa réalisation. En effet, le temps de réalisation d'une interprétation peut être important, il faut donc vérifier que celle-ci est nécessaire avant sa mise en place.

Afin de simplifier le développement de nouvelles interprétations, notre outil propose un cadre de travail facilitant l'enrichissement de l'outil. Aucune contrainte technique n'est définie sur le développement d'interprétations. L'approche proposée par l'IDM présente un intérêt certain en répondant à la majorité de nos attentes que sont des outils de transformation/génération simples et puissants. De plus, le choix de cette approche permet de rester cohérent avec l'outil de modélisation réalisé à l'aide de modèles.

Comme nous pouvons le voir sur la Figure 0-8, le mécanisme d'interprétation se décompose en trois étapes. Ce mécanisme prend en entrée un modèle de spécification des besoins fonctionnels conforme au langage proposé précédemment. La première étape est une transformation de modèles qui permet de réaliser l'interprétation du besoin et donc concentre toute son intelligence. Cette transformation de modèles est contrainte, dans notre système, à prendre un unique modèle en entrée conforme à notre langage et à retourner un unique modèle en sortie. Le méta-modèle cible est spécifique à l'interprétation. Il peut être relatif à une carte conceptuelle, à une documentation, à la modélisation UML, à l'évaluation des risques… La seconde étape est l'étape de génération. Chaque artefact de génération est appliqué sur le modèle issu de la transformation. Dans le cas des langages de génération à base de *templates*, l'artefact de génération sera un *template* utilisant une grammaire spécifique en fonction du langage utilisé. Puis une étape supplémentaire est exécutée afin de réaliser des opérations difficilement faisables par transformation ou génération. Ce mécanisme est entièrement automatisé et accessible à l'utilisateur à travers l'interface de l'outil de modélisation. Une

interprétation est donc structurée sous la forme d'un triplet composé d'une transformation de modèles associée au méta-modèle cible correspondant (carte conceptuelle, documentation, etc.), d'un ensemble d'artefacts de génération s'appliquant sur le méta-modèle précédent puis d'un ensemble de scripts.

Afin d'outiller ce mécanisme, nous avons utilisé des technologies adaptées à chacune des étapes. En ce qui concerne la transformation de modèles, nous proposons d'utiliser une transformation conforme au langage ATL [JOU 05] présenté dans le chapitre précédent. A la suite de celle-ci, un langage de génération à base de *templates* est utilisé à l'aide de la technologie Acceleo[18]. Pour finir, la phase de post traitement est réalisée à l'aide de scripts ANT[19]. Ces technologies ne sont qu'une instanciation de notre mécanisme d'interprétation. D'autres combinaisons peuvent être proposées mais celle-ci présente les caractéristiques attendues et s'intègre parfaitement bien dans la plateforme Eclipse.

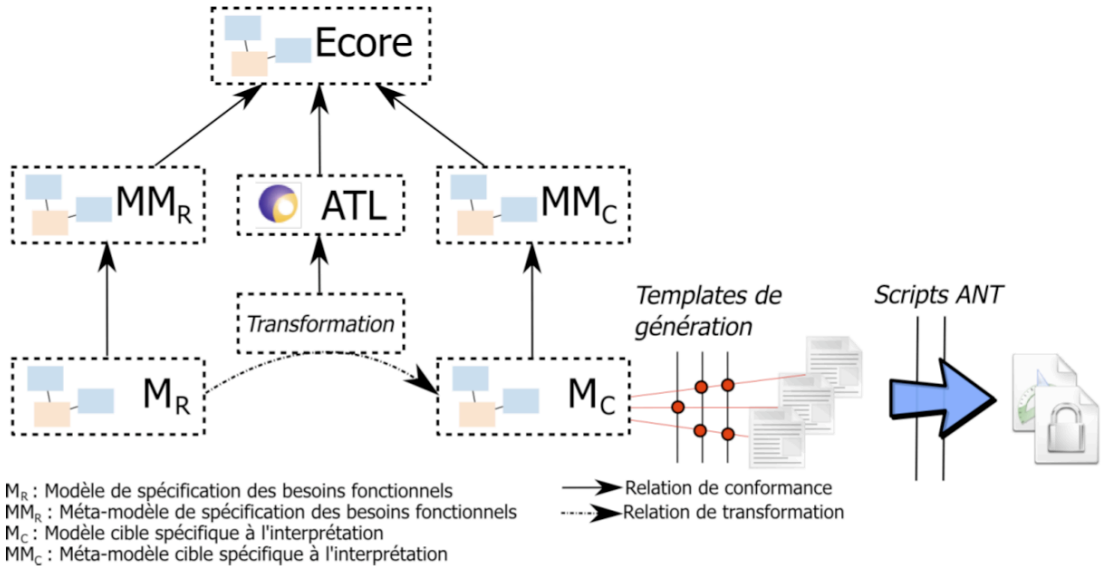


Figure 0-8 : Mécanisme d'interprétation

[18] http://www.acceleo.org
[19] http://ant.apache.org

C. Positionnement de la méthodologie SUN dans le processus de conception

Les méthodes de conception de SII proposent des démarches de gestion des cycles de développement (cf. chapitre A). L'objectif de notre approche est de proposer un cadre méthodologique et des outils afin de formaliser et valider la spécification d'un SII. Cette démarche doit être indépendante de l'approche de conception choisie, à l'instar de la conception centrée utilisateur. Les phases décrites dans la suite représentent les phases menant aux spécifications fonctionnelles. On suppose que le recueil initial des besoins a déjà été effectué avant d'initialiser cette méthode. Nous obtenons, en sortie de celle-ci, une spécification claire des besoins fonctionnels sous un format permettant la liaison avec le processus de conception qui suit. Ainsi dans le cycle en cascade, cette approche s'insèrera dans les étapes d'étude de faisabilité, d'analyse des besoins et de conception (cf. Figure 0-9). Tandis que ce seront plutôt les étapes d'analyse des besoins, d'élaboration de la recette, de spécification et de tests qui seront concernées dans le cycle en V (cf. Figure 0-10).

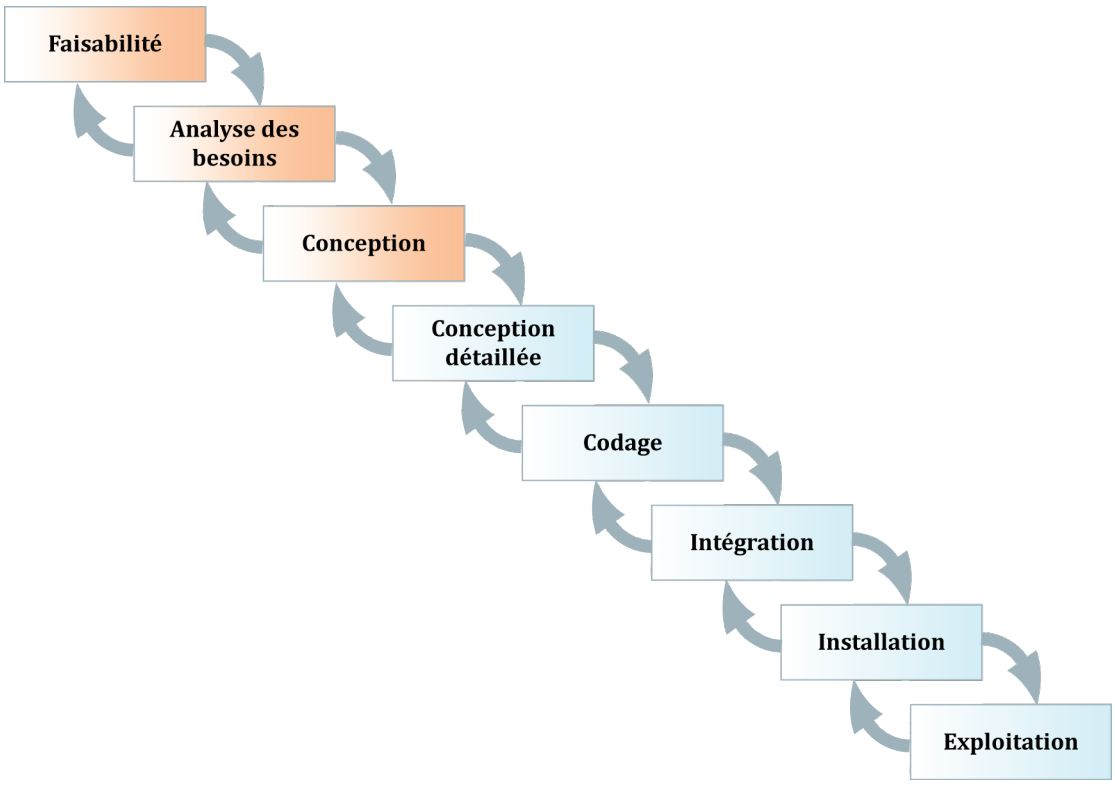


Figure 0-9 : Positionnement de SUN dans un cycle en cascade

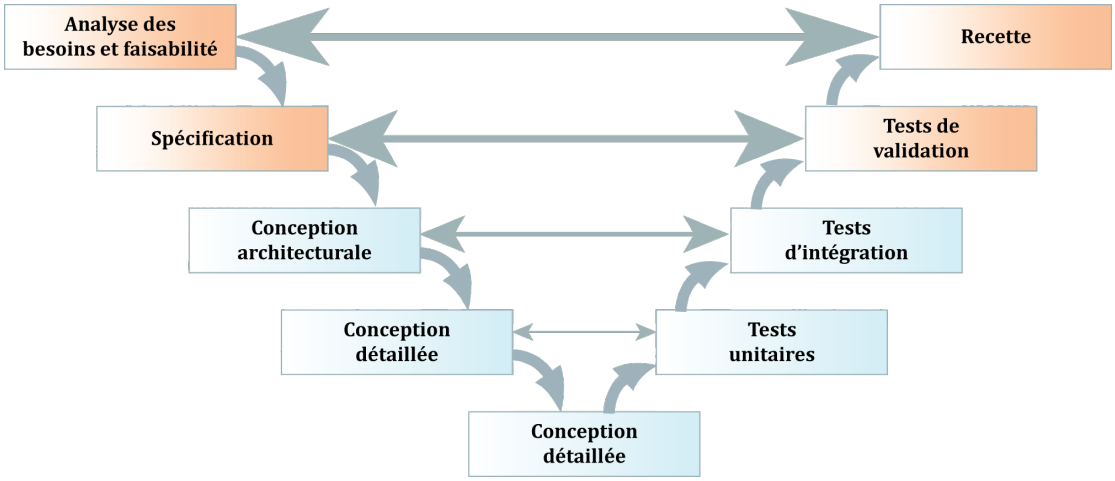


Figure 0-10 : Positionnement de SUN dans un cycle en V

Nous avons pu voir dans le premier chapitre que le prototypage permet non seulement de valider les spécifications fonctionnelles mais c'est également un moyen de faire émerger de nouveaux besoins. Nous préconisons donc d'utiliser

cette approche dans des cycles itératifs tels que le cycle en spirale ou les méthodes agiles afin d'optimiser le retour de connaissances sur le SII.

D. Les acteurs de la méthodologie SUN

L'analyse des besoins et la spécification fonctionnelle implique quatre types de participants :

a. L'utilisateur final : il représente les futurs utilisateurs du système en cours de conception. Afin d'assurer une meilleure acceptation du nouvel outil, l'hétérogénéité des caractéristiques liées à ce type d'acteur doit être considérée.
b. Le responsable métier : il représente le gestionnaire du projet du côté client. Ce profil pose le cahier des charges et les apports technologiques attendus en accord avec l'expert métier. Il est le seul à mesurer les contraintes et limites inhérentes à l'organisation ainsi que les choix stratégiques de l'organisation.
c. L'expert métier : il possède une compétence forte sur l'activité supportée par le futur système. Il fait office de représentant des utilisateurs finaux.
d. Le concepteur : il représente la composante technique dans le processus de développement. Il a en charge de comprendre les besoins du client et d'en définir des spécifications permettant l'implémentation d'un système répondant à ce besoin.

A la différence des méthodes présentées ci-dessus, la méthodologie SUN permet de faire intervenir de nombreux participants sur des supports de communication variés et adaptés à chacun. Après avoir défini les différents types de participants, il s'agit de définir et d'outiller leurs activités et leurs interactions dans le processus de conception. Chaque participant génère des connaissances qui sont nécessaires à capturer et capitaliser.

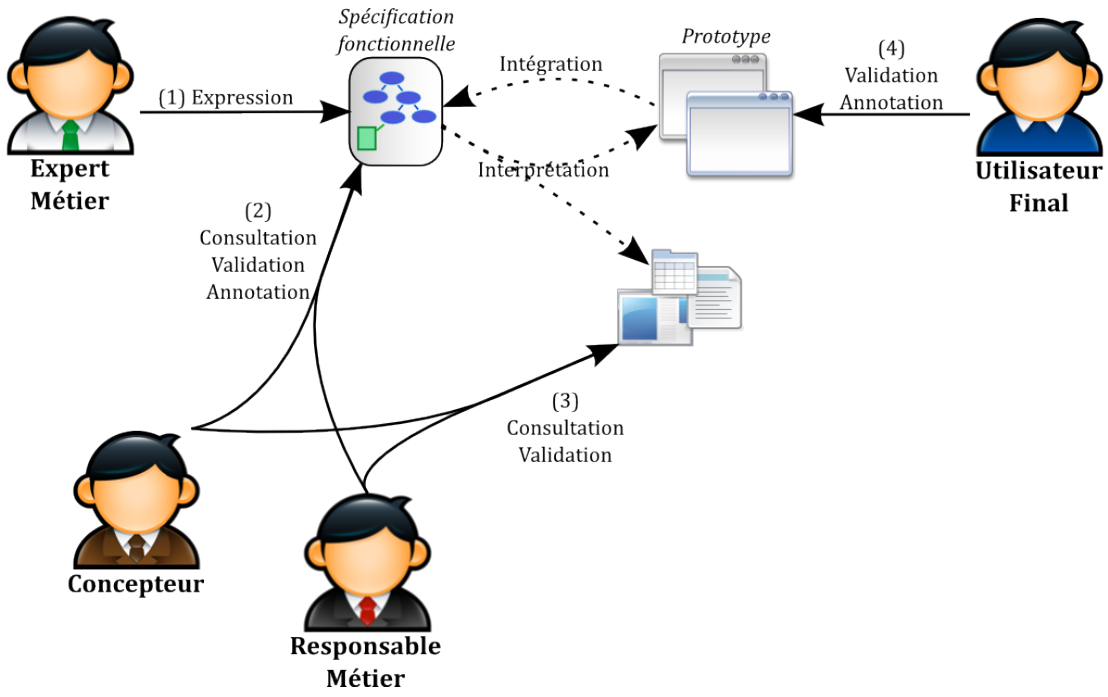


Figure 0-11 : Schéma de communication de la méthode SUN

Le premier mouvement représente le flux de définition des besoins, celui-ci est initialisé et principalement géré par l'expert métier (cf. [1] Figure 0-11). Il a la possibilité de consulter le concepteur ou le responsable métier pour orienter la définition des besoins selon des contraintes qu'il ne pourrait mesurer telles les contraintes techniques ou organisationnelles par exemple. Dans ce processus de définition, le concepteur et le responsable ont évidemment la possibilité de consulter la spécification mais aussi de l'annoter (cf. [2] Figure 0-11). Ce mécanisme permettra ensuite de suivre les évolutions de la définition des besoins.

Le second mouvement représente le flux de validation des besoins. Celui-ci est destiné aux trois autres profils : utilisateur final, concepteur, responsable métier. Le concepteur et le responsable métier ont la possibilité d'intervenir directement sur la spécification fonctionnelle par un mécanisme d'annotation ou en interagissant avec l'expert métier après consultation du résultat des interprétations (cf. [3] Figure 0-11). Les interprétations utilisées peuvent être diverses et fonction de l'étape de conception et du profil concerné. Cette phase de définition et de validation des besoins aboutit à la spécification fonctionnelle d'un premier prototype de SII. Celui-ci servira de base à la validation par les

utilisateurs finaux. Ceux-ci pourront ainsi consulter, utiliser et enfin annoter ce prototype pour décrire des problèmes, des incompréhensions ou des oublis (cf. [4] Figure 0-11). Ces annotations seront ensuite réintégrées automatiquement dans le modèle de spécification du besoin. Ce mécanisme peut faire penser à des rapports de bugs ou de nouvelles fonctionnalités. Celui-ci est entièrement automatisé et centralisé dans la spécification des besoins.

E. Étapes de SUN

Le point d'entrée de la méthode SUN est le recueil initial des besoins. Cette étape n'est pas couverte par la méthode mais quelques préconisations sont faites. Elle consiste en une analyse de l'existant et des évolutions souhaitées. Elle s'appuiera donc sur une analyse précise des interfaces existantes, permettant d'extraire le squelette du dictionnaire de données, des documents déjà rédigées par le client pour définir les évolutions souhaitées par rapport à un système existant ou à des référentiels de confiance. Suite à ce recueil, une première version de la spécification fonctionnelle peut être réalisée.

1. Structuration des besoins utilisateur

Suite au recueil initial du besoin, il est possible de définir la hiérarchie des objectifs. Par cette structuration, nous allons définir, dans ses grandes lignes, le périmètre fonctionnel du futur système. L'arbre sera décomposé jusqu'au moment où un objectif correspondra à une tâche concrète. Celui-ci sera principalement rédigé en accord avec le responsable métier, chargé de garantir que le système sera en accord avec les objectifs stratégiques de l'organisation.

Idéalement, chaque objectif sera documenté et accompagné d'un synopsis. Ce synopsis permettra ensuite de rédiger des cahiers de recette et de tests plus facilement. Bien évidemment, ces deux informations ne pourront pas être saisies à chaque fois. La définition d'un synopsis sur un objectif sera principalement utilisée dans le cas des objectifs « feuille », c'est-à-dire lorsque l'objectif correspondra à une tâche.

De plus, chaque objectif sera priorisé par l'expert métier, chargé d'identifier les tâches récurrentes donc primordiales, et le responsable métier, chargé

d'identifier les activités à fort retour sur investissement. Cette information permettra ensuite au concepteur de hiérarchiser le processus de développement.

Aucune obligation n'est faite dans la méthode SUN pour valider le besoin. Toutefois, quelques préconisations sont faites. Afin de valider la structuration des besoins, il est possible d'utiliser des cartes conceptuelles permettant l'organisation et la représentation des connaissances. Nous reviendrons plus longuement sur cette interprétation dans le chapitre suivant.

2. Définition du dictionnaire

En parallèle de la structuration des besoins utilisateurs, il est possible de définir le dictionnaire de données du futur système. Celui-ci permet de définir les concepts métier et les liens entre eux. Les entités, les attributs et les relations permettront de capitaliser cette connaissance sur le dictionnaire de données.

Cette étape peut être fastidieuse mais des mécanismes automatiques peuvent être imaginés en fonction du contexte du projet. Tout d'abord, dans le cas de la modernisation d'un système, il est possible d'obtenir automatiquement le dictionnaire, à partir par exemple, d'une base de données. Toutefois, ce dictionnaire devra être raffiné à cause des différences qui peuvent subvenir entre les concepts métier et leurs images dans une base de données.

De plus, comme pour l'étape de structuration des besoins, il est préconisé d'utiliser des méthodes issues de l'analyse de l'existant. L'étude d'interfaces existantes (du système implanté ou de systèmes concurrents) ou de référentiels internes à l'organisation ou issus de normes permettra également d'extraire la majeure partie du dictionnaire de données.

Cette étape est primordiale et devra être correctement validée par le concepteur et le responsable métier. Le concepteur sera chargé de valider l'accessibilité actuelle ou future à l'ensemble des informations définies dans le dictionnaire de données. Le responsable métier devra vérifier que le périmètre des informations manipulées dans le futur système est en accord, encore une fois, avec les objectifs stratégiques de l'organisation. Ce périmètre sera, toutefois, vérifié une nouvelle fois dans les étapes suivantes.

Afin de valider cette partie, l'utilisation de cartes conceptuelles peut être utilisées. Toutefois, dans le cas de dictionnaire avec beaucoup de relations, la carte conceptuelle deviendra rapidement illisible et la visualisation native sous forme de modèle E-A sera plus performante.

3. Analyse de la structure et de la stratégie de l'organisation

Cette étape consiste à définir les différents acteurs du futur système ainsi que leurs responsabilités. Chaque acteur humain fera office de profil utilisateur dans le futur système. Il sera également possible de définir des acteurs non humains tels que des automates.

La définition des responsabilités est très importante. Elle consiste à spécifier les tâches associées à chacun des acteurs. Cette partie peut faire l'objet de discussions importantes, accentuées si la mise en place d'un nouveau système est accompagnée par une restructuration de l'organisation.

Les cartes conceptuelles peuvent, une nouvelle fois, être utilisées pour valider cette partie à la nuance près qu'elles seront centrées sur les acteurs. Nous pouvons définir une carte conceptuelle par acteur avec l'ensemble des objectifs qui sont sous sa responsabilité. Le responsable métier sera chargé de valider cette partie en vérifiant la cohérence avec les profils de poste existant dans l'organisation.

Une autre partie devra être définie dans cette étape : il s'agit des processus métier. Chacun d'entre eux sera défini en spécifiant une suite d'objectifs qui sera ensuite validée par le responsable métier et l'utilisateur final à travers le prototype généré. Cette partie sera plus longuement étudiée par la suite.

4. Spécification des politiques d'accès aux données

La spécification des politiques d'accès est le chaînon manquant permettant de lier les objectifs utilisateurs et le dictionnaire de données. Elle consiste à définir quelles informations vont être consommées et produites pour chacune des tâches. Nous définissons donc pour chaque tâche un chemin dans le dictionnaire de données dont le point d'entrée sera une entité. Ce chemin se compose ensuite d'un parcours à travers les relations dans le dictionnaire de données. Différents

droits seront appliqués sur les éléments du dictionnaire de données en utilisant les quatre opérations de base fournies par un CRUD (« *Create, Read, Update, Delete* ») : la création, la lecture, la mise à jour et la suppression.

Les opérations de création et de suppression seront réservées aux entités. Les droits de lecture et de mise à jour pourront être employées sur l'ensemble des concepts du dictionnaire de données, c'est-à-dire les entités, les attributs ainsi que les associations.

Les politiques d'accès ainsi définies nous permettront de proposer des interfaces adaptées à chacune des activités. Les interfaces seront ensuite générées puis validées par les utilisateurs finaux à travers le prototype généré. L'interprétation permettant de générer un prototype est la seule à être obligatoirement utilisée par la méthode. Il est primordial, selon notre méthode, de proposer un processus de prototypage automatique et incrémental afin d'impliquer les utilisateurs finaux dans le processus de conception.

5. Validation\Annotation par les utilisateurs

Cette étape se concentre sur la validation des étapes précédentes. A travers le prototype généré par l'une des interprétations présentées dans le chapitre suivant, l'utilisateur final pourra vérifier :

a. L'ensemble des tâches sous la responsabilité d'un profil utilisateur : il pourra ainsi agir si certaines tâches ont été oubliées ou mal affectées ;
b. Pour chaque tâche :
    i. L'accès à l'ensemble des informations nécessaires à la réalisation de celle-ci ;
    ii. L'impossibilité d'accéder à des informations inutiles à la réalisation de celle-ci ;
    iii. Le moyen de renseigner l'ensemble des informations produites pendant ou après celle-ci ;
    iv. L'impossibilité de renseigner des informations qui ne peuvent être produites pendant ou après celle-ci.

L'utilisateur final peut, directement, annoter la spécification fonctionnelle à travers le prototype. L'ensemble des annotations est ensuite réintégré à l'outil de modélisation puis analysé par l'expert métier.

F. Synthèse

Dans ce chapitre, nous avons présenté une méthode générique de spécification de SII présentée sur la Figure 0-12. Celle-ci commence à l'issu du recueil initial des besoins. La spécification des besoins fonctionnels sous la responsabilité de l'expert métier est validée par le concepteur et/ou le responsable métier en fonction du degré de raffinement de la spécification. Cette validation se fera, soit par consultation et annotation directement sur la spécification utilisant le langage proposé, soit par consultation d'une image de celle-ci obtenue automatiquement à l'aide d'une interprétation. Si de nouvelles annotations ont émergé, l'expert métier devra proposer une nouvelle version de sa spécification. Dans le cas contraire, un prototype sera généré automatiquement et présenté à l'utilisateur final. Celui-ci pourra utiliser l'outil et émettre, lui aussi, des annotations qui seront automatiquement réintégrées à la spécification. Dans le cas où de nouvelles annotations ont été définies, le processus recommence et l'expert métier devra proposer une nouvelle version si les annotations le justifient. Sinon, nous sortons alors du processus de spécification. Des interprétations pourront être utilisées pour exporter les connaissances accumulées durant ce processus telle qu'une transformation vers des modèles conceptuels.

Cette méthode générique s'applique à tout cycle de conception basé sur les modèles classiques. Cependant, nous préconisons dans notre méthode l'utilisation du prototypage pour la validation des besoins, il est donc logique et cohérent de l'utiliser sur des méthodes utilisant les cycles de conception incrémentale avec une validation par les utilisateurs régulière et fréquente.

Dans la suite de ce mémoire, nous allons présenter les différentes interprétations utilisables dans la majorité des projets de développement de SII, et plus particulièrement celles évoquées dans ce chapitre. Nous présenterons ensuite la

mise en œuvre de cette méthode sur un exemple concret de systèmes d'organisation de conférence.

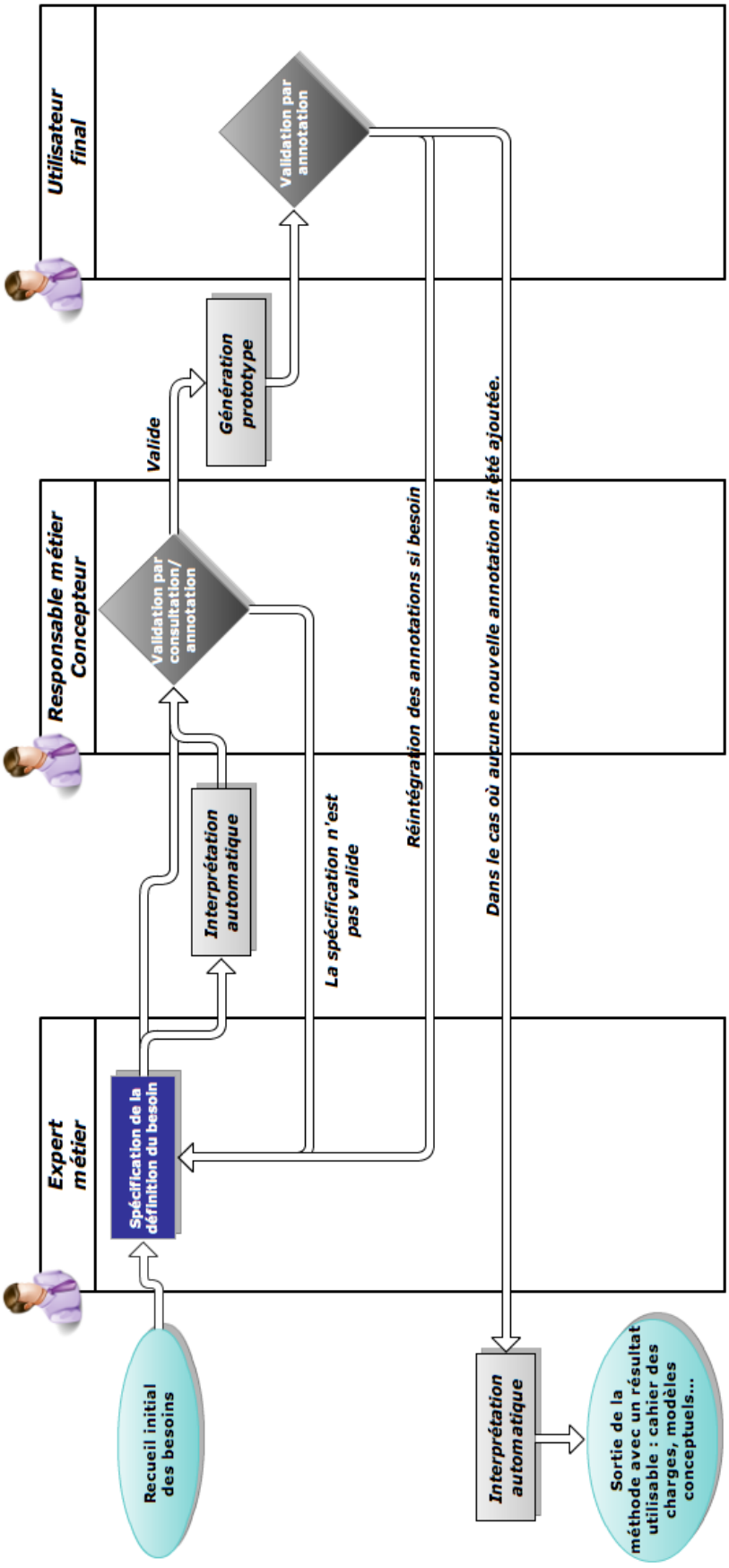


Figure 0-12 : Déroulement de la méthode SUN

Interprétations disponibles dans SUN

Le mécanisme d'interprétation proposé dans cette méthodologie permet de proposer un autre point de vue sur la spécification du besoin exprimée à l'aide du langage proposé. Cet objectif permet de cacher la complexité du langage et de l'outil qui l'accompagne. Nous supposons que les acteurs de cette méthodologie seront beaucoup plus sensibles au contenu de la spécification si la forme leur est familière.

L'aspect important du mécanisme d'interprétation est son automatisation. Il est primordial qu'il ne soit en aucun cas consommateur de temps, hormis évidemment durant sa phase de conception. Nous présenterons dans ce chapitre les interprétations que nous avons réalisées et que nous mettons à disposition dans SUN. Celles-ci utilisent l'IDM et chacune des étapes sera présentée précisément.

Pour la présentation de ces différentes interprétations, nous présenterons à chaque fois le méta-modèle en utilisant le formalisme UML, la transformation de modèles présentée schématiquement ainsi que les *templates* de génération.

A. Identification des anomalies

La première interprétation qui vient à l'esprit est de vérifier qu'il n'existe pas d'anomalie dans la spécification du besoin. Nous cherchons à vérifier ici la cohérence de la spécification. Nous vérifierons, par exemple :

a. Que le nom ainsi que la documentation sont renseignés ;
b. Qu'un agent a au moins un objectif sous sa responsabilité ;
c. Qu'il n'existe pas de redondance dans les responsabilités, c'est-à-dire qu'un agent ne peut être responsable d'un objectif et d'un de ses enfants ;
d. Qu'il n'existe aucune politique d'accès pour un objectif ;
e. Qu'une entité, un attribut ou une relation sont référencés dans au moins une politique d'accès ;
f. etc.

1. Méta-modèle

Le méta-modèle du diagnostic est extrêmement simple : la racine de celui-ci est représentée par un *Diagnostic* composé de problèmes (*Problem*). Chaque problème ou anomalie est défini par une sévérité (erreur, critique et avertissement), une description correspondant à l'anomalie puis deux propriétés permettant d'identifier l'élément à l'aide de son type et son nom. Les anomalies critiques doivent impérativement être corrigées sinon les interprétations risquent de ne pas pouvoir aboutir tandis que celles correspondant à une erreur doivent être corrigées mais n'empêchent pas l'exécution des interprétations. Toutefois, le résultat de celles-ci peut être incohérent.

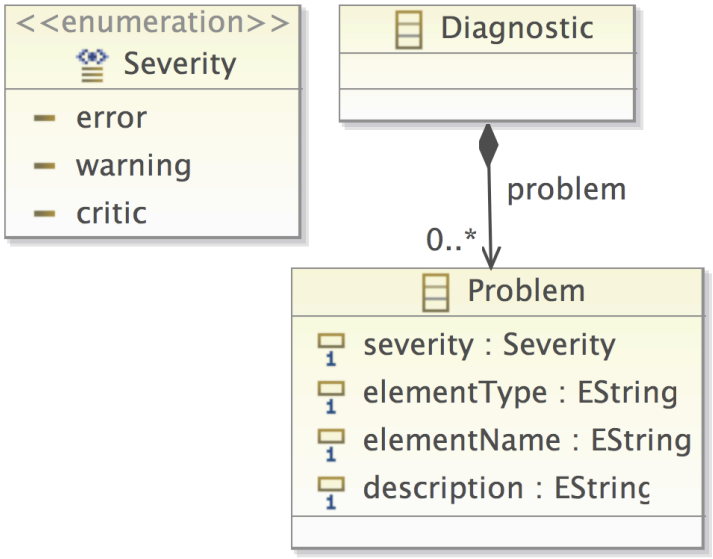


Figure 0-1 : Méta-modèle du diagnostic

2. Transformation de modèles

La transformation de modèles (cf. Listing 6 / B.Annexe C) est composée d'une règle (*Diagnostic*) permettant de créer un objet à partir d'une définition des besoins (*RequirementsDefinition*). Ensuite, pour chacun des éléments principaux du méta-modèle, c'est-à-dire les agents (*Agent*), les objectifs (*Goal*), les entités (*Entity*), les attributs (*Attribute*) et les associations (*RelationShip*) ainsi que l'élément parent (*BasicElement*), nous réalisons toute une suite de vérification que nous insérons dans la partie concernant les problèmes du diagnostique. Pour chacun des éléments principaux, il existe une méthode chargée d'appeler les méthodes spécifiques pour chaque type. Chacune d'elle peut ensuite générer

une anomalie s'il y a lieu. Sur la Figure 0-2, on peut voir que chaque élément du modèle est passé dans un crible spécifique à son type. Ensuite, l'ensemble des sous méthodes sont appelées afin d'exécuter les vérifications. Certaines méthodes génèreront une anomalie de sévérité différente. Dans le cas où aucun problème n'a été identifié, la méthode ne génère aucun élément et la méthode suivante est appelée.

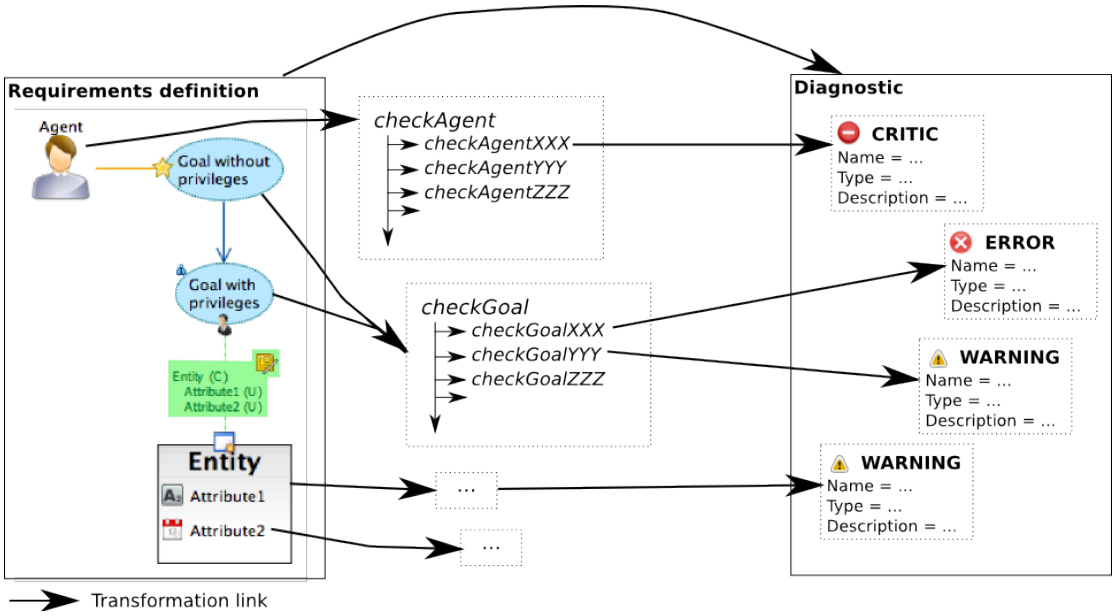


Figure 0-2 : Schéma de la transformation vers un diagnostique

3. Génération

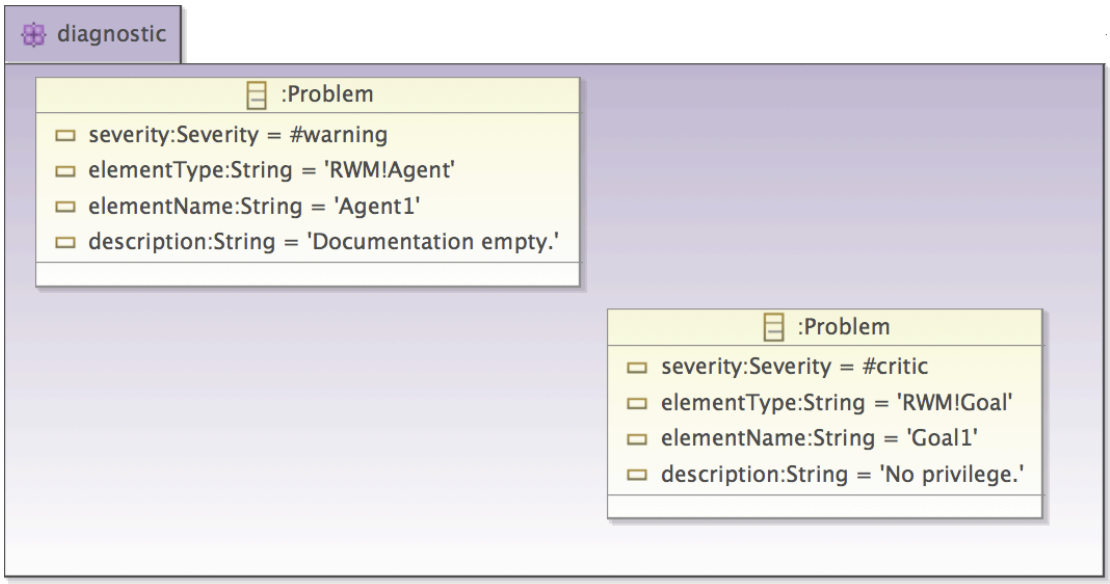


Figure 0-3 : Diagramme d'objets du modèle de diagnostique

Le modèle obtenu (cf. Figure 0-3), représenté à l'aide d'un diagramme d'objets UML, est composé d'un diagnostic lui-même composé d'un ensemble d'anomalies. Celui-ci doit être généré pour être interprétable par un outil. Nous décidons de proposer notre propre interpréteur qui sera ensuite intégré aux autres outils web proposés par les différentes interprétations. Pour faire simple, nous décidons de générer un fichier JSON[20] listant l'ensemble des anomalies. Celui-ci sera ensuite utilisé par notre interpréteur.

```
<%script type="Diagnostic.Diagnostic" name="Problem"
file="webtool/data/analysis/problem.json"%>
{
       "diagnostic":[
       <%for (problem){%>
       {
              "type":"<%elementType%>",
              "name":"<%elementName%>",
              "severity":"<%severity%>",
              "description":"<%description%>"
        },
       <%}%>
       ]
}
```

Listing 1 : Template Acceleo de génération à base de diagnostique

Pour l'ensemble des anomalies listées dans le diagnostique, nous générons une portion de JSON définissant le type de l'élément, son nom, la sévérité de l'anomalie ainsi que la description de celle-ci. Le résultat obtenu sur un exemple correspond au code présenté dans le Listing 2.

```
{
   "diagnostic":[
                 {
                "type":"RWM!Agent",
                "name":"Agent1",
                "severity":"warning",
```

[20] JSON (JavaScript Object Notation) est un format de données textuel, générique, dérivé de la notation des objets du langage ECMAScript. Il permet de représenter de l'information structurée.

```
                "description":"Documentation empty."
        },
        {
                "type":"RWM!Goal",
                "name":"Goal1",
                "severity":"critic",
                "description":"No privilege."
        },
        ...
    ]
}
```

Listing 2 : Fichier JSON généré à partir d'un modèle de diagnostique

Ce fichier JSON est ensuite interprété puis utilisé par l'outil web que nous avons développé permettant ainsi une meilleure visibilité.

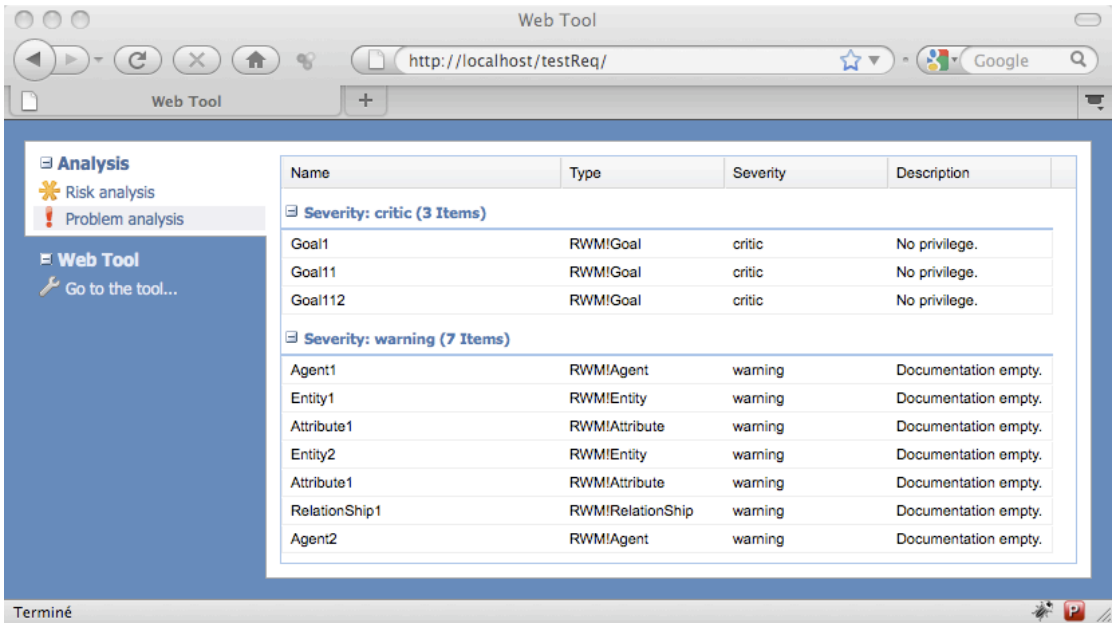


Figure 0-4 : Diagnostique interprété sous la forme d'un tableau

B. Approche par les cartes heuristiques ou « mind maps »

Le *Mind Mapping* est une technique graphique d'organisation de l'information. En français, nous parlerons de « cartes heuristiques » ou de « cartes mentales ». Les cartes heuristiques sont utilisées afin d'apprendre, améliorer sa mémoire, réaliser des brainstormings, penser visuellement et résoudre toutes sortes de problème.

Cette méthode a été modélisée par un psychologue anglais, Tony Buzan, dans les années 1970. En français, vous la retrouverez sous différentes appellations: carte heuristique, schéma heuristique, topogramme, carte des idées, carte mentale, arbre à idées, etc. Concrètement, il s'agit d'un diagramme qui représente un ensemble de relations entre des données, suivant une architecture arborescente. Son intérêt réside dans sa capacité à structurer et à faire émerger de l'information. Il s'agit donc d'un outil de visualisation qui participe à la prise de notes et à la cartographie de l'information.

Le véritable atout de la carte heuristique est de permettre à son utilisateur de se concentrer sur certains détails particuliers, tout en conservant une appréciation globale de la situation. Elle permet ainsi de comprendre de manière plus efficace et plus rapide des situations complexes. Pour résumer le *Mind Mapping* consiste à créer une interface aussi simple et efficace que possible pour exprimer ses idées de manière aussi structurée que possible.

1. Méta-modèle

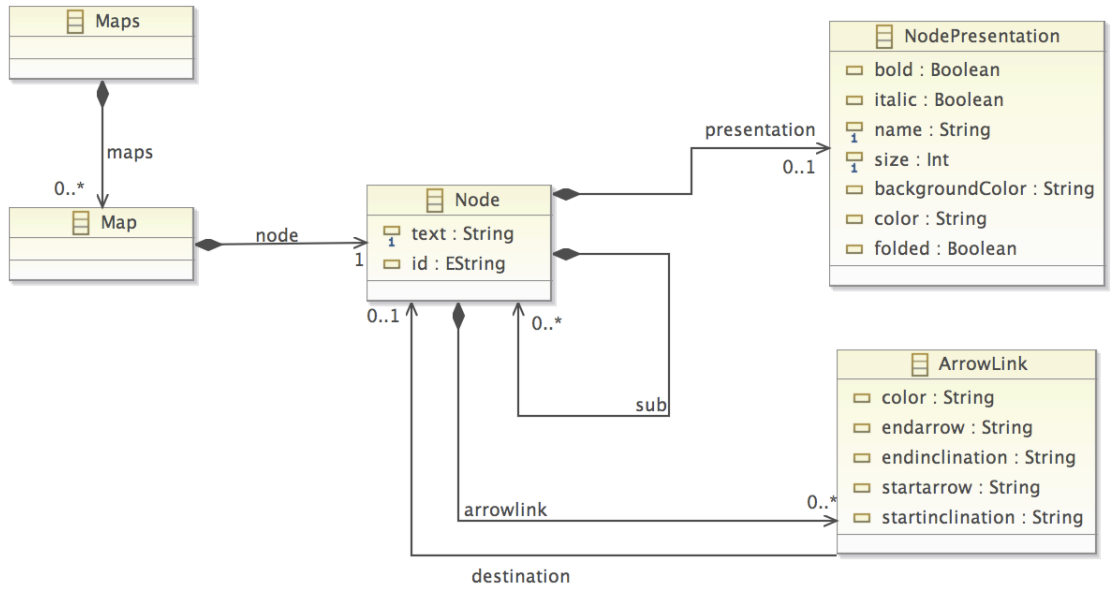


Figure 0-5 : Méta-modèle de cartes heuristiques

Un modèle de cartes heuristiques (cf. Figure 0-5) est composé de plusieurs cartes (*Map*) composées d'un nœud principal (*Node*) correspondant à la racine de l'arbre. Celles-ci sont regroupées dans un seul objet (*Maps*). Un nœud peut être décomposé en sous nœud. Il est également possible de créer des liens

transverses (*ArrowLink*) entre nœuds. La présentation de chaque nœud peut ensuite être personnalisable (*NodePresentation*).

Nous avons choisi de définir plusieurs cartes dans un seul modèle en raison des contraintes apportées par le mécanisme d'interprétation. Celui-ci impose d'obtenir un seul modèle à la sortie de la transformation. Or il est possible d'obtenir plusieurs cartes comme dans le cas où nous obtenons une carte des objectifs pour chacun des agents définis dans la spécification des besoins. Il nous faut donc toutes les définir dans un seul modèle.

2. Transformation de modèles

Plusieurs sous interprétations sont disponibles autour des cartes heuristiques. L'intérêt de l'usage de carte heuristique est de pouvoir *zoomer* sur une partie de la spécification du besoin fonctionnel. Deux types de découpage sont mis en place :

a. Un découpage par axe, c'est-à-dire en se focalisant sur un sous-ensemble des concepts mis à disposition au travers du langage : nous retrouvons une carte heuristique représentant la structure d'information dans laquelle nous ne visualisons que l'axe concernant la définition du domaine, et une autre carte représentant l'arbre des objectifs dans laquelle nous ne visualisons qu'une partie de l'axe concernant la définition des activités utilisateur.

b. Un découpage par agent, c'est-à-dire que nous allons créer une carte heuristique en se focalisant, soit sur la structure d'information, soit sur les objectifs sous leur responsabilité. La carte concernant la structure d'information par agent permet de vérifier la cohérence entre les polices d'accès définies et les droits d'accès réels. La carte concernant les objectifs permet de vérifier que les activités définies dans la spécification du besoin sont en adéquation avec la répartition des tâches actuelle ou souhaitée.

Nous ne présenterons précisément qu'une transformation par découpage : l'arbre des objectifs (cf. Listing 7 / Annexe C) ainsi que la liste des objectifs par

agent (cf. Listing 8/ Annexe C). Pour la première carte, nous devons obtenir une seule carte par définition des besoins dans laquelle nous retrouvons un nœud par objectif défini, conformément à la décomposition faite dans la définition. Dans le cas où une documentation a été ajoutée à un objectif, nous devons la présenter comme sous nœud mais avec une présentation différente.

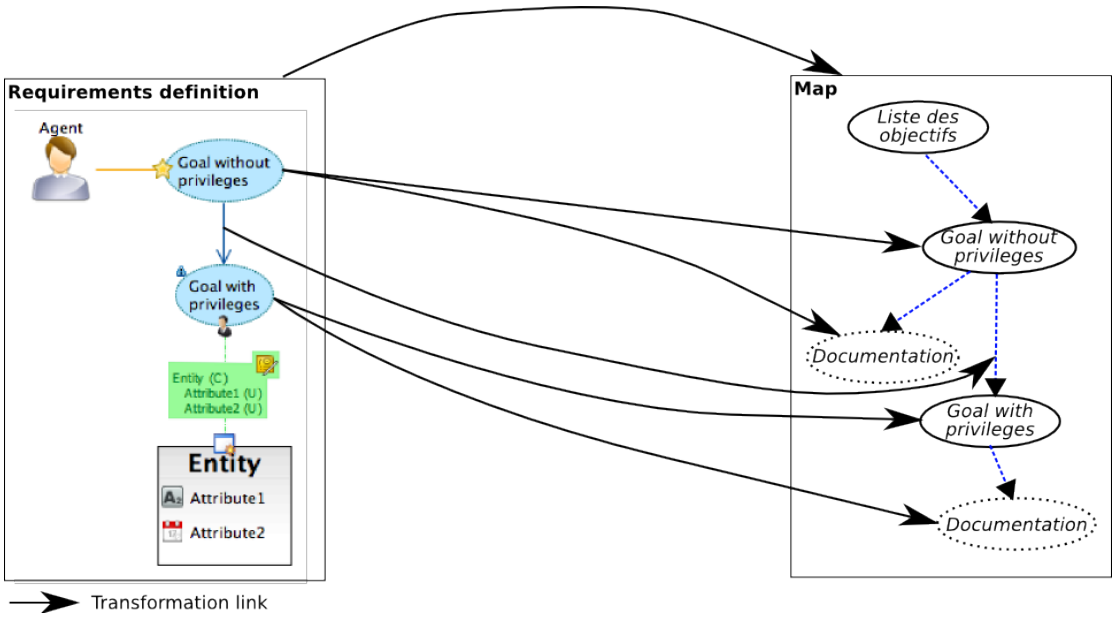


Figure 0-6 : Schéma de la transformation ATL Req2GoalList

La règle « *définition* » permet de créer une carte heuristique avec un nœud principal que nous appelons « *Liste des objectifs* ». A ce nœud, nous ajoutons comme enfant l'ensemble des nœuds correspondants à des objectifs non contenus par d'autres objectifs. Il existe ensuite deux règles pour les objectifs :

- « *goal* » permettant de définir un nœud par objectif et
- « *goalWithDocumentation* » étendant la première et permettant de définir un nœud par objectif mais également un sous nœud contenant la documentation de celui-ci.

Nous retrouverons sur la Figure 0-7 un exemple de modèle obtenu à la sortie de la transformation sous la forme d'un diagramme d'objets. Nous pouvons y voir une carte composée d'un nœud central lui même décomposé en sous nœuds. Certains d'entre eux incluent une présentation particulière avec un objet de type *NodePresentation*.

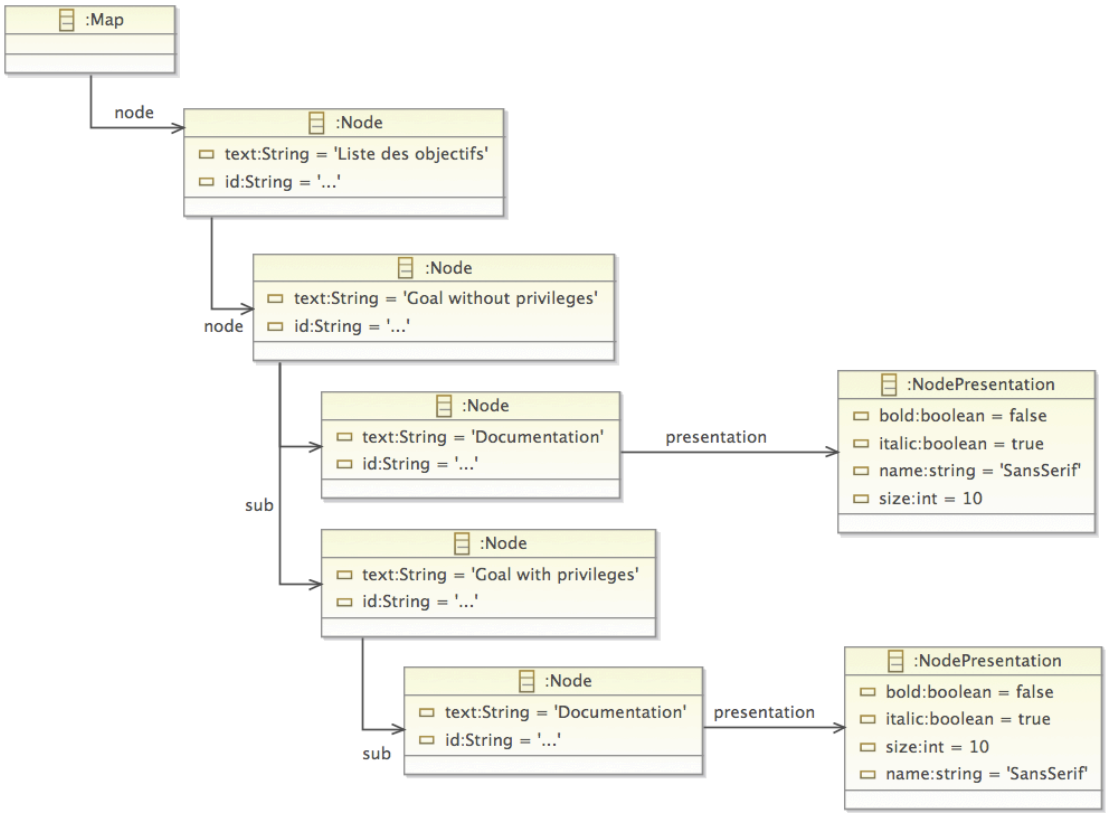


Figure 0-7 : Diagramme d'objets du modèle de carte conceptuelle issue de la transformation Req2GoalList

La transformation, permettant d'obtenir une carte par agent, est légèrement différente (cf. Figure 0-8). Tout d'abord, nous devons générer plusieurs cartes regroupées dans un objet « *Maps* ». Chaque agent est à l'origine d'une nouvelle carte, composée d'un nœud « *Objectifs* » ainsi que l'ensemble des sous nœuds sous leur responsabilité. La différence majeure est que la création d'un nœud en accord avec un objectif ne se fait plus à l'aide d'une règle de correspondance mais à l'aide d'une règle impérative. Cette différence existe parce qu'un objectif peut être sous la responsabilité de plusieurs acteurs donc présent dans plusieurs cartes heuristiques.

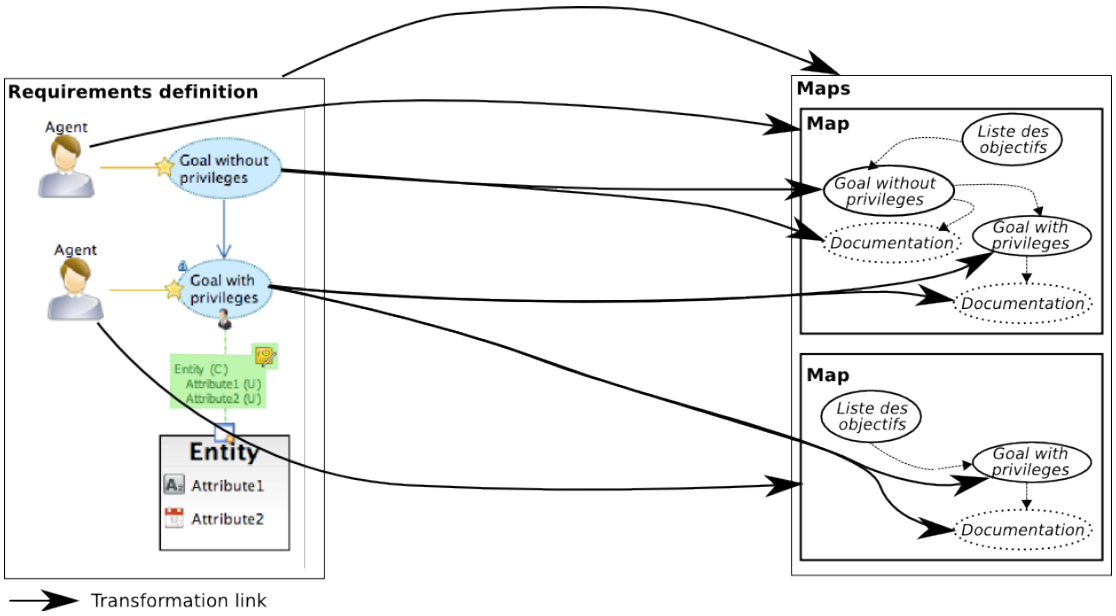


Figure 0-8 : Schéma de la transformation ATL Req2GoalListByAgent

3. Génération

Après avoir obtenu un modèle de carte heuristique indépendant de l'outil à l'issu de la transformation de modèles, il faut désormais définir les *templates* de génération. Le choix de l'outil technologique doit se faire dans cette étape. Dans le cadre de cette thèse, nous faisons le choix de s'orienter vers une solution gratuite. Parmi celles-ci, nous retrouveons XMind©, Mindomo©, FreeMind©... Nous sélectionnons le dernier en raison de sa popularité. Le format d'entrée de celui est un fichier XML conforme à un schéma[21]. Celui-ci nous a aidé à construire le fichier attendu par FreeMind.

```
<%script type="MindMap.Map" name="Map" file="<%node.text%>.mm"%>
<?xml version="1.0" encoding="ISO-8859-1"?>
<map version="1.0">
  <%node.PrintNode%>
</map>
<%script type="MindMap.Node" name="PrintNode"%>
<node text="<%text%>" folded="<%presentation.folded%>" id="<%id%>">
  <%for (sub){%>
    <%PrintNode%>
```

[21] http://freemind.cvs.sourceforge.net/viewvc/freemind/freemind/freemind.xsd?view=markup

```
  <%}%>
  <%for (presentation){%>
    <%PrintPresentation%>
  <%}%>
  <%for (arrowlink){%>
    <%PrintArrowLink%>
  <%}%>
</node>
<%script type="MindMap.NodePresentation" name="PrintPresentation"%>
<font bold="<%bold%>" italic="<%italic%>" size="<%size%>" name="<%name%>"/>
<%script type="MindMap.ArrowLink" name="PrintArrowLink"%>
<arrowlink startarrow="<%startarrow%>" endarrow="<%endarrow%>"
startinclination="<%startinclination%>"  endinclination="<%endinclination%>"
destination="<%destination.id%>" color="<%color%>"/>
```

Listing 3 : Template de génération à base de carte heuristique

Le premier script permet de créer la base du fichier XML avec le nœud principal. Ensuite, la portion XML de chaque nœud est générée. Une dernière partie concernant les liens entre nœuds est présentée mais celle-ci n'est pas utilisée dans les deux exemples présentés. Le résultat obtenu est présenté sur la Figure 0-9.

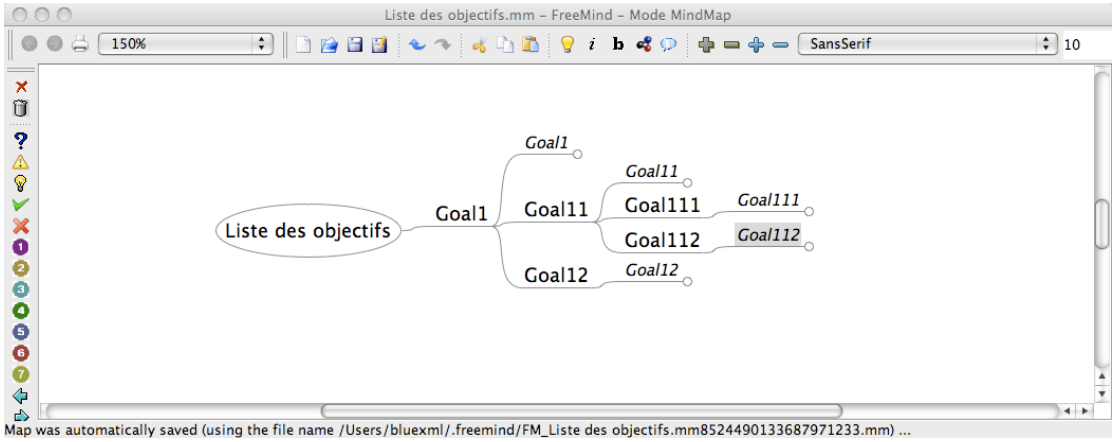


Figure 0-9 : Résultat de la génération ouvert avec l'outil FreeMind

C. Évaluation des risques

L'estimation des coûts dans le processus de conception d'un logiciel correspond à la prédiction des efforts requis pour développer celui-ci. Une estimation juste est bien évidemment primordiale pour les développeurs et les clients. Elles

peuvent être utilisées pour les appels d'offre, les négociations contractuelles, la planification, le suivi et le contrôle. Sous-estimer le coût d'un logiciel peut avoir beaucoup de conséquences dans le processus de développement dont la livraison d'un outil avec des fonctionnalités peu développées, de pauvre qualité, contenant des bugs [LEU 02]. Les sociétés informatiques rencontrent aujourd'hui souvent des problèmes pour développer des systèmes incluant l'ensemble des fonctionnalités prévues au lancement du projet. La raison principale est une réelle difficulté de prévoir le temps et les ressources nécessaires à l'exécution d'une tâche. C'est pour cela qu'il est intéressant d'identifier quelles sont les principales fonctionnalités requises et nécessitant plus de temps et plus de réflexion que les autres. Donner une réponse appropriée à ce sous-ensemble d'objectifs devient un levier d'action non négligeable dans la satisfaction du client face à l'outil livré et donc influence le succès d'un projet. Cette interprétation vise à identifier ces objectifs majeurs, que nous avons définis sous le terme d'objectif à haut risque.

Différents paramètres peuvent entrer dans une formule de calcul de risque tels que le coût de réalisation, le temps nécessaire à sa réalisation, le niveau de sécurité requis, le niveau de performance attendu, la maturité des technologies, les compétences nécessaires, etc. Nous limitons le calcul de cette interprétation aux caractéristiques déductibles de notre langage, c'est-à-dire la difficulté de réalisation par rapport à l'impact sur le succès du projet global. Dans cette interprétation, nous ne cherchons pas à quantifier le risque mais à réaliser une comparaison entre les différents objectifs. La valeur calculée nous permet ainsi de hiérarchiser les objectifs afin de donner une priorité aux objectifs de haute importance. Un objectif est considéré à haut risque :

a. S'il est déclaré comme important par le client qui a lui-même fixé cette valeur;
b. S'il est sous la responsabilité de plusieurs agents : si la fonctionnalité développée, correspondant à l'objectif, est utilisée par la majorité des utilisateurs, alors il est important de fournir une solution adaptée afin de réduire la résistance au changement et donc augmenter l'acceptation du nouvel outil ;

c. S'il est lié à un grand volume d'information, c'est-à-dire si sa politique d'accès permet de visualiser/modifier une grande quantité d'information : si l'agent doit produire ou accéder à de nombreuses informations pour achever une tâche, alors il est primordial d'apporter une très grande attention au développement des interfaces afin de présenter de la meilleur manière possible les données et ainsi réduire sa complexité.

Le risque d'un objectif est calculé en faisant la moyenne entre la priorité définie par le client (*priority*), la couverture nécessaire sur le futur système d'information (*ISCoverage*) ainsi que le pourcentage d'agent responsable de celui-ci (*UserCoverage*) :

$$\forall g \in G, risk(g) = \frac{priority(g) + ISCoverage(g) + UserCoverage(g)}{3} \quad (1)$$

Où *G* représente l'ensemble des objectifs.

Pour calculer la couverture sur le système d'information, nous faisons la moyenne entre le pourcentage d'entités référencées et le pourcentage d'attributs référencés. Le pourcentage d'entités référencées est la division entre le nombre d'entités référencées (en utilisant *accessEntities*) et le nombre total d'entités. Le pourcentage d'attributs référencés est la division entre le nombre d'attributs référencés (en utilisant *accessAttributes*) et le nombre total d'attributs contenus par des entités accessibles par la même politique d'accès :

$$\forall g \in G,$$
$$ISCoverage(g) = \frac{\left(\frac{|accessEntities(g)|}{|E|} + \frac{|accessAttributes(g)|}{|a \in A, entity(a) \in accessEntities(g)|} \right)}{2} \quad (2)$$

Où *G* représente l'ensemble des objectifs ;

E représente l'ensemble des entités ;

A représente l'ensemble des attributs.

Enfin, le pourcentage d'agent est calculé simplement en divisant le nombre d'agents responsables (en utilisant *responsible*) par le nombre total d'agents :

$$\forall g \in G, UserCoverage(g) = \frac{|responsible(g)|}{|AG|} \quad (3)$$

Où *G* représente l'ensemble des objectifs ;

AG représente l'ensemble des agents.

Nous calculons également le risque pour chaque agent. Celui-ci permet d'identifier les futurs acteurs majeurs du SII. Pour cela, nous divisons l'ensemble des objectifs et ses sous-objectifs sous sa responsabilité (en utilisant *goals_under_responsibility*) par l'ensemble des objectifs de la spécification :

$$\forall a \in AG, Coverage(a) = \frac{|goals\_under\_responsibility(a)|}{|G|} \quad (4)$$

Où *AG* représente l'ensemble des agents ;

G représente l'ensemble des objectifs.

De même pour les entités, ce risque permet d'identifier les concepts majeurs et ainsi il est important de bien les valider avec l'ensemble des participants du processus de conception. Pour calculer celui-ci, nous divisons la quantité de privilèges définis sur elle-même et ses attributs (en utilisant *privileges*) par le nombre total de privilèges :

$$\forall e \in E, risk(e) = \frac{|privileges(e)|}{|P|} \quad (5)$$

Où *E* représente l'ensemble des entités ;

P représente l'ensemble des privilèges.

1. Méta-modèle

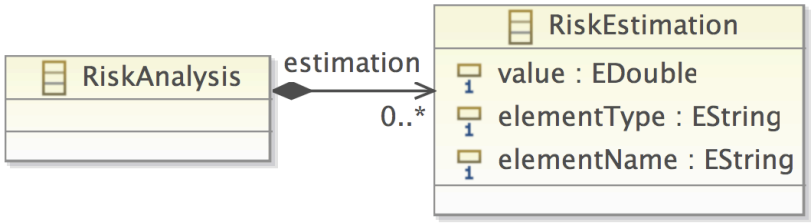


Figure 0-10 : Méta-modèle de calcul de risque

Le méta-modèle est extrêmement simple : il est composé d'un élément racine (*RiskAnalysis*) composé d'un ensemble d'estimation. Chacune de celle-ci est caractérisée par une valeur représentant le risque ainsi que deux propriétés pour caractériser un élément par son type et son nom.

2. Transformation de modèles

Pour chacun des types étudiés, c'est-à-dire les objectifs, les entités et les attributs, nous créons une nouvelle estimation en appelant la méthode permettant de calculer le risque (cf. Listing 9/ B.Annexe C). Nous pouvons voir sur la Figure 0-11 les différentes estimations obtenues à partir des différents types en entrée.

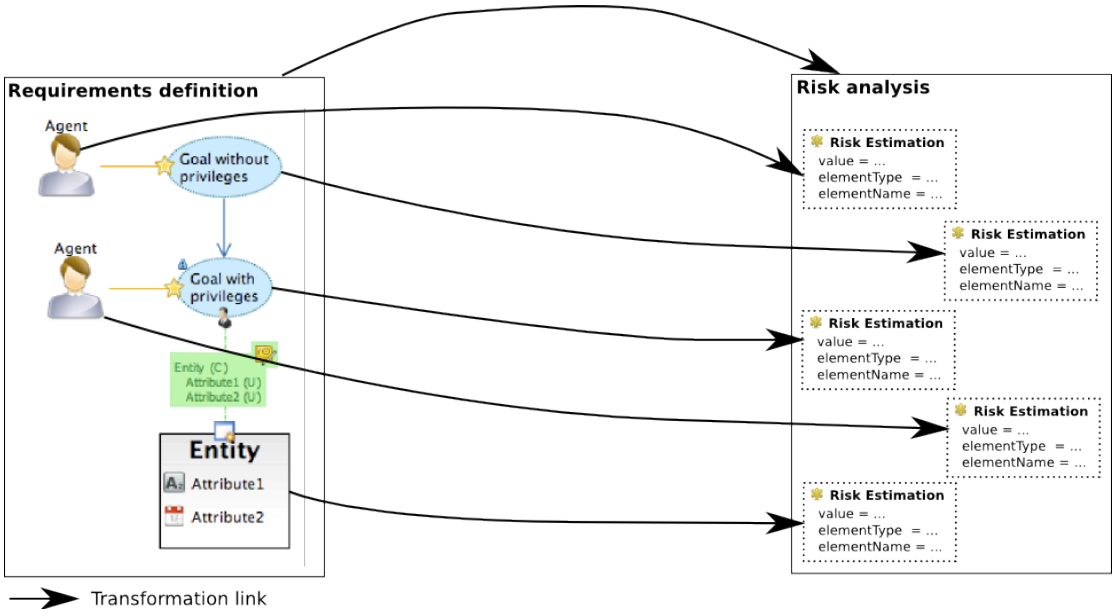


Figure 0-11: Schéma de la transformation ATL Req2Risk

3. Génération

Le modèle obtenu composé d'un diagnostic lui-même composé d'un ensemble d'estimation doit être généré pour être interprétable par un outil. À l'instar de l'interprétation permettant de vérifier les anomalies, nous décidons de proposer notre propre interpréteur qui sera ensuite intégré aux autres outils web proposés par les différentes interprétations en générant un fichier JSON.

```
<%script type="Risk.Diagnostic" name="Diagnostic"
file="webtool/data/analysis/diagnostic.json"%>
```

```
{
       "diagnostic":[
      <%for (estimation){%>
       {
              "type":"<%elementType%>",
              "name":"<%elementName%>",
              "value":"<%value%>"
        },
       ]
}
```

Listing 4 : Template de génération à base d'une estimation des risques

Le résultat obtenu est ensuite utilisé par notre interpréteur (cf. Figure 0-12) afin de visualiser l'ensemble des estimations. Une colonne avec une barre de progression a été utilisée afin de visualiser plus simplement les éléments stratégiques. On peut y voir que plus la valeur est importante, plus le risque est fort. On doit considérer les éléments à haut risque avec le plus grand soin car une réponse adaptée à ces éléments cristallise le succès du projet entier. Il est important de noter qu'il est peu intéressant de comparer les valeurs de risque entre deux éléments de deux types différents à cause de leur formule de calcul divergente. Ce qu'il faut comprendre de cette figure est que l'équipe de conception devra impérativement apporter une réponse adaptée aux besoins liés à l'activité définie dans l'objectif « Goal112 ». Les autres objectifs sont à traiter avec la même priorité. Cette information est intéressante dans le cas où les délais et/ou budgets ont été dépassés. Cela permet d'identifier ainsi les tâches sur lesquelles les efforts de développement devront être concnetrés.

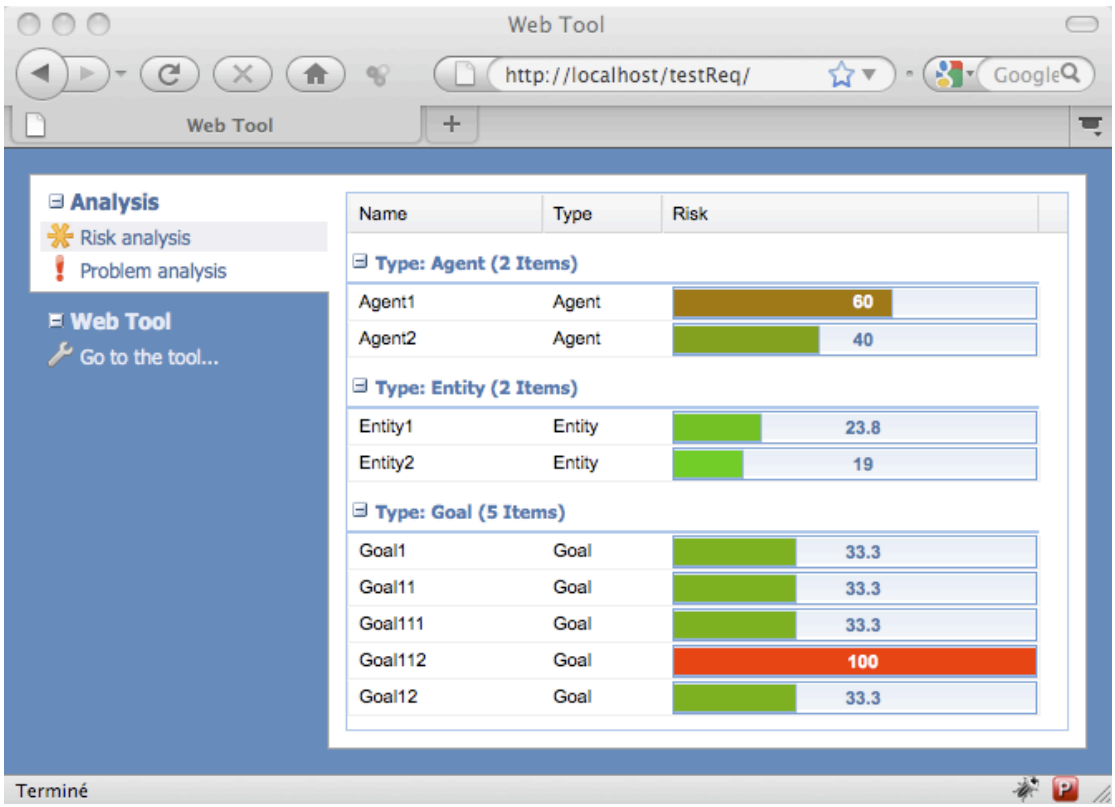


Figure 0-12 : Estimation des risques obtenue après génération

D. Prototypage

Généralement, il est difficile de communiquer efficacement avec les experts fonctionnels ou les futurs utilisateurs d'un SII. Ils rencontrent généralement des problèmes pour imaginer et critiquer un document représentant le SII final. Un moyen efficace pour illustrer la spécification fonctionnelle est l'utilisation de prototypes. Les utilisateurs peuvent directement interagir avec l'outil généré. Cependant, le problème principal est d'expliquer clairement que le prototype n'est utilisé que pour évaluer le moyen proposé pour consommer et produire de l'information. L'interface utilisateur n'est donc pas couverte par cette évaluation. Il est utilisé pour vérifier qu'un utilisateur est capable d'obtenir les informations nécessaires à la réalisation de son activité et de renseigner les informations produites pendant l'exécution de sa tâche. Pour résumer, il est utile pour vérifier la politique d'accès de chacun des objectifs.

Nous avons donc fait le choix de construire un outil web accessible par n'importe quel navigateur afin d'en simplifier son usage. Le prototype web se décompose en trois parties identifiables sur la Figure 0-13 :

a. Gestion des données utilisée pour la configuration d'une base de données (partie bleue). Nous faisons le choix d'implémenter une base de données relationnelle. Nous définissons donc trois éléments : un schéma (*Schema*) composé d'un ensemble de tables (*Table*) caractérisées par un ensemble de champs (*Field*).
b. Gestion de la structure générale de la page (partie violette) afin de décomposer la page en plusieurs colonnes (*FramePage*).
c. Gestion de la connexion et de l'authentification (partie rouge). Nous définissons une page de connexion (*LoginPage*) avec un ensemble de liens pointant vers d'autres pages. Il n'y a donc pas vraiment d'authentification à proprement parler car l'utilisateur ne renseigne jamais de nom d'utilisateur et de mot de passe. Les liens définis dans cette page permettent d'accéder à des pages composées (*FramePage*) d'autres pages incluant une page avec la liste des objectifs (*GoalPage*) spécifique à un agent.
d. Accès aux fonctionnalités (partie verte) afin de lister (*GoalPage*) un arbre d'objectifs (*GoalItem*) permettant d'accéder à d'autres pages telles que celles correspondantes aux objectifs incluant une politique d'accès (*DataPage*);
e. Ensemble des pages concernant une fonctionnalité permettant de créer, lire, modifier ou supprimer de l'information (partie orange). Chaque page (*DataPage*) est composée d'un ensemble de composants (*Component*) liés à une table de la base de données. Chaque composant inclut un ensemble de propriétés qui peuvent correspondre à un champ de la table (*ComponentAttribute*) ou à une navigation vers une autre table (*ComponentRelationShip*). En fonction des droits définis, le projet web devra les bonnes actions pour la consultation, la modification, la création et la suppression.

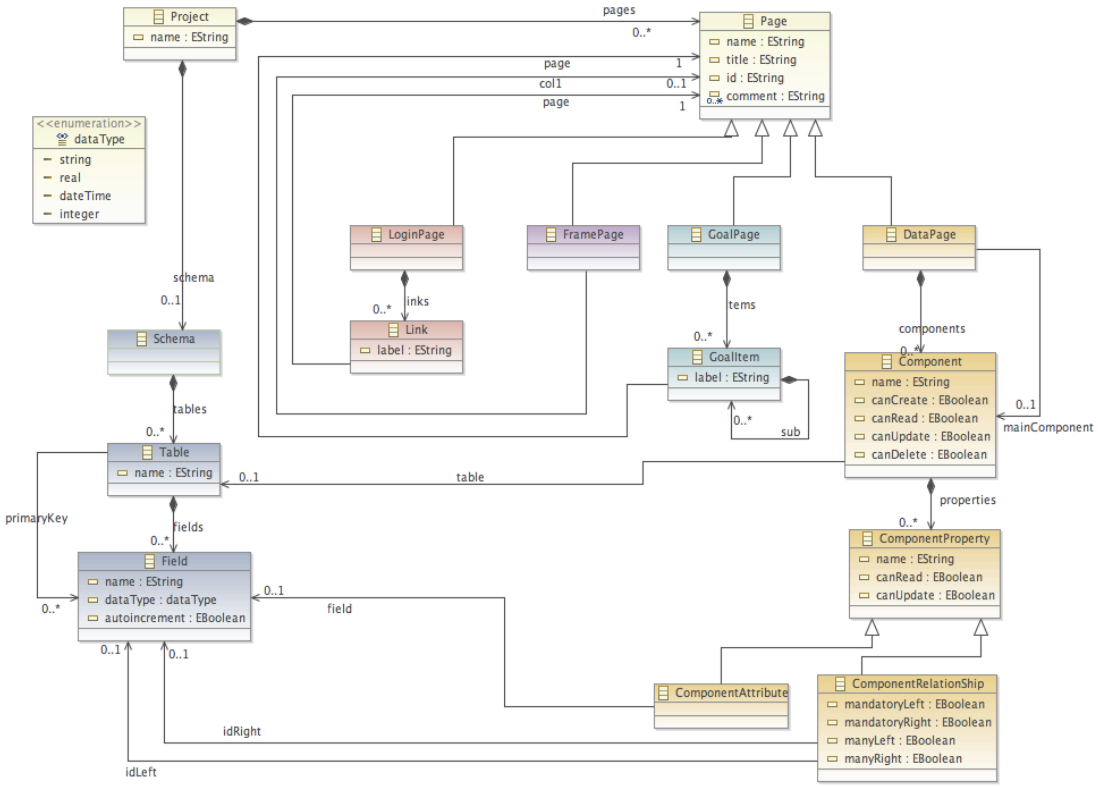


Figure 0-13 : Méta-modèle de projet Web

Pendant la transformation de modèles (cf. Listing 10/ B.Annexe C), le prototype web et le schéma de la base de données sont initialisés directement à partir de la spécification des besoins. Au même moment, la page de connexion est créé et enrichi de liens pour accéder aux pages contenant les objectifs de chacun des agents. Ces pages référencent d'autres pages qui sont généralement le moyen d'accéder à la fonctionnalité finale permettant de réaliser l'ensemble des actions permises par la police d'accès définie sur l'objectif (cf. Figure 0-14).

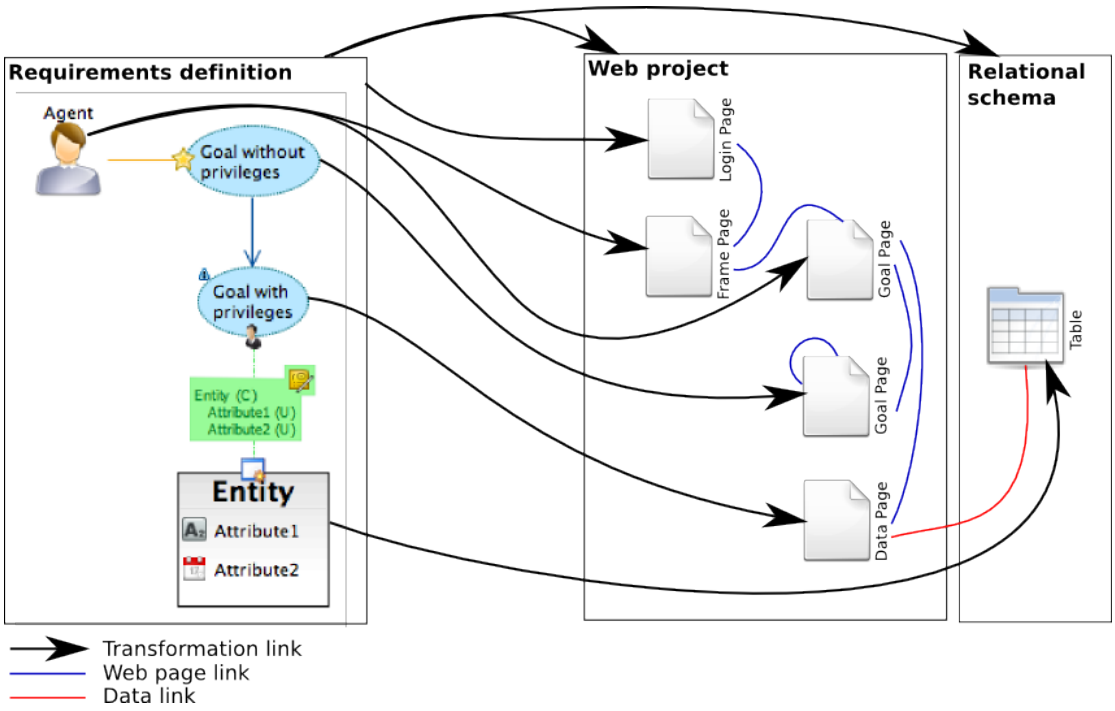


Figure 0-14 : Illustration de la transformation pour générer un prototype

Pendant la phase de génération, nous appliquons un ensemble de templates permettant de configurer le squelette du prototype Web. Ceux-ci nous permettent de générer principalement des pages PHP[22] et HTML[23]. Un script SQL[24] est également généré afin de configurer la base de données. Les utilisateurs peuvent ainsi utiliser l'outil et le critiquer directement.

[22] PHP (Hypertext Preprocessor) est un langage de scripts libre principalement utilisé pour produire des pages web dynamiques à partir d'une base de données par exemple.
[23] HTML (Hypertext Markup Language) est le format de données conçu pour représenter les pages web.
[24] SQL (Structured Query Language) est un langage informatique normalisé qui sert à demander des opérations sur des bases de données. La partie langage de manipulation de données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données.

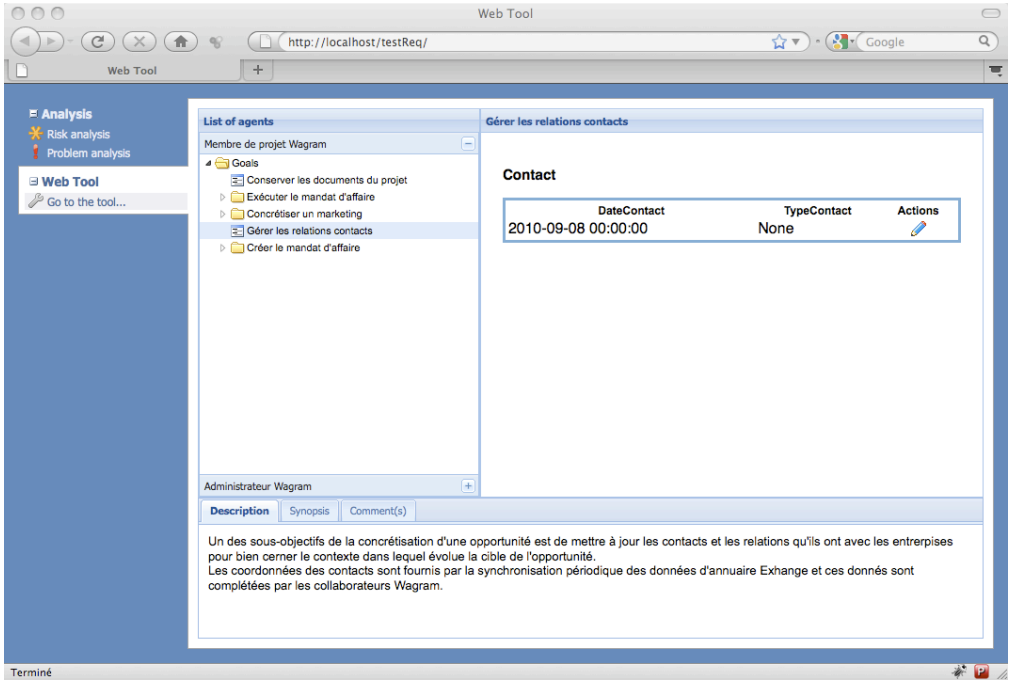


Figure 0-15 : Prototype Web généré

E. Documentation

Malgré une proposition de langage pour spécifier les besoins fonctionnels d'un SII, la référence est tout de même un document écrit afin, entre autre, de poser les bases à une négociation contractuelle. Ce document sera principalement utilisé à la fin du processus de spécification et reprendra l'ensemble des informations définies dans la spécification.

L'objectif principal de cette interprétation est d'obtenir un document modifiable avec un éditeur de texte traditionnel permettant ainsi de communiquer plus facilement à l'issu de la phase de spécification du besoin. Ce document pourra être enrichi par exemple par les exigences non fonctionnelles telles que la sécurité, la performance…

1. Méta-modèle

Le méta-modèle de documentation (cf. Figure 0-16) est inspiré du langage *DocBook*[25]. L'élément principal permet de définir un document (*Book*) composé d'un ensemble de sections. Chacune d'entre-elles peut ensuite se décomposer en sous-section et paragraphe. Celui-ci se décompose ensuite en différentes valeurs (*ParagraphValue*) qui peuvent être un bloc de texte (*TextualValue*), un bloc de texte mis en valeur (*EmphasisValue*), un tableau (*InformalTableValue*), une liste de valeurs (*ItemizedListValue*) ou encore un lien vers une section (*XRefValue*).

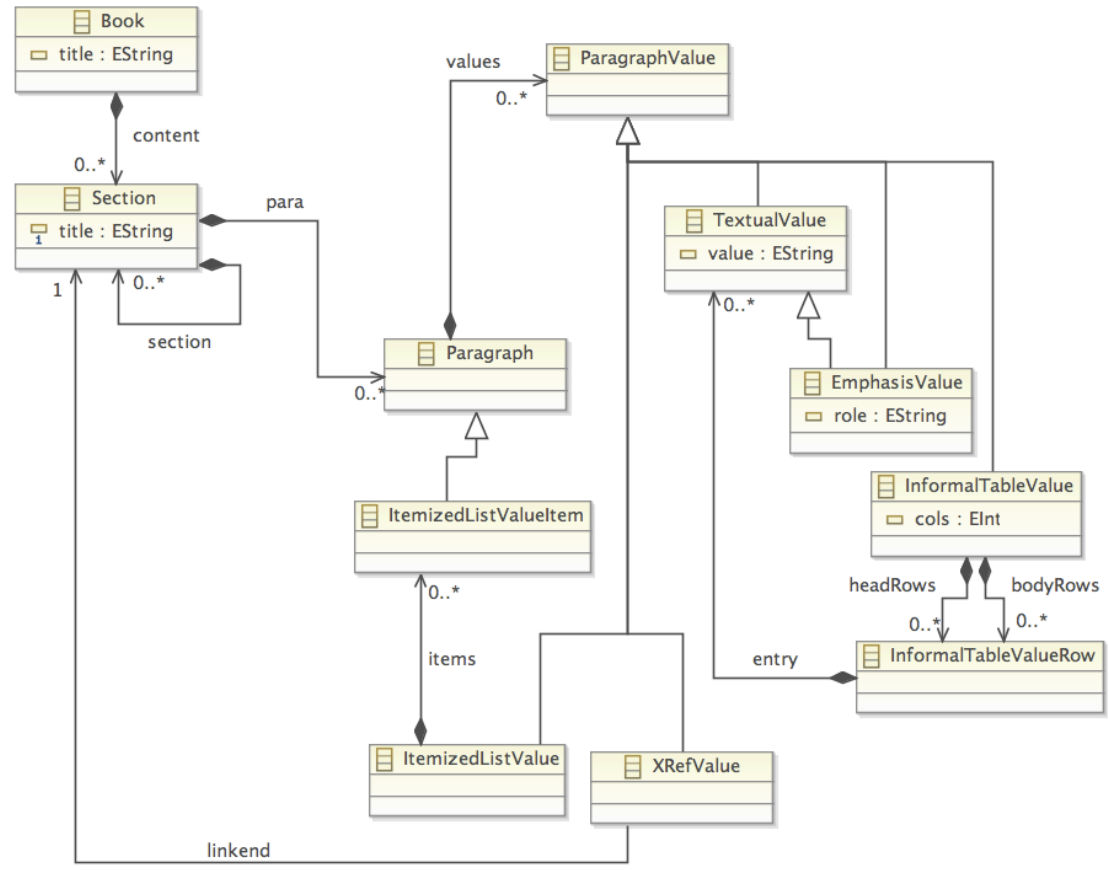


Figure 0-16 : Méta-modèle de documentation

[25] *DocBook* est un langage permettant l'écriture de n'importe quel type de document, bien qu'il s'agisse en général de documents techniques. À la différence d'autres, il n'est pas conçu pour spécifier la forme sous laquelle le document va être présenté, mais uniquement pour spécifier sa structure. Son principal avantage est d'être un langage très fortement structuré avec de nombreuses balises et des règles d'imbrications strictes.

2. Transformation de modèles

Pour des raisons de clarté, nous ne présenterons pas l'ensemble de la transformation (cf. Listing 11/ B.Annexe C) qui est relativement complexe et nous nous concentrons sur la règle permettant de générer une section par objectif. Celle-ci est composée de quatre sous-sections (description, synopsis, agents responsables et politique d'accès). La première sous-section permet d'afficher la documentation saisie par l'utilisateur dans la spécification. Pour cela, nous ajoutons un bloc de texte à cette section de type *TextualValue* contenant le texte de la documentation.

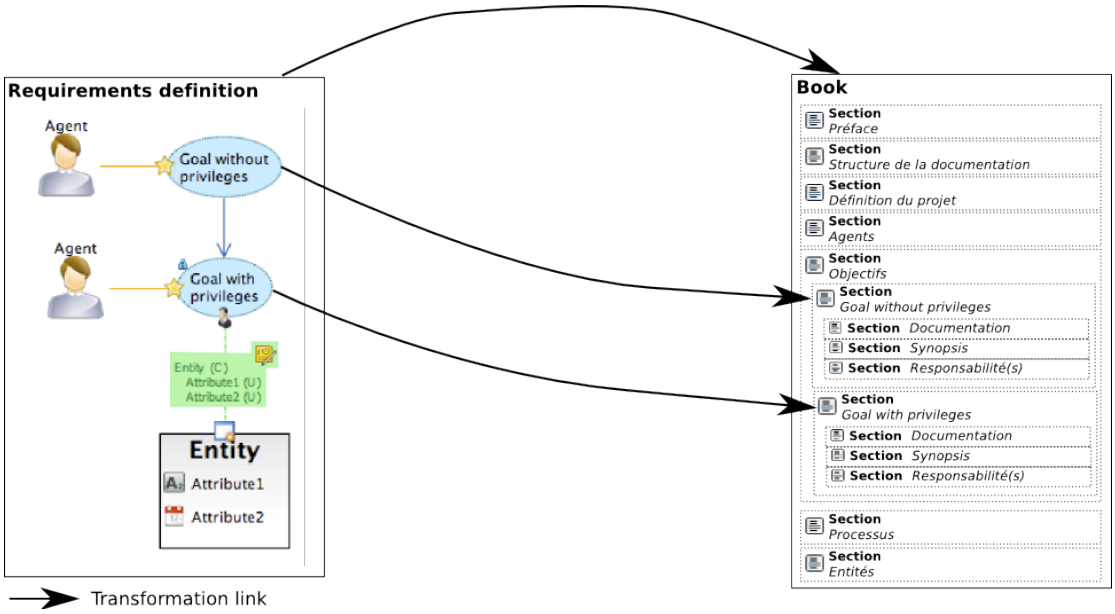


Figure 0-17: Schéma de la transformation ATL Req2Documentation

A l'issu de la transformation, nous obtenons un modèle de documentation (cf. Figure 0-18) composée d'une arborescence de sections et paragraphes. Ce modèle servira de base à la génération du document final. La décomposition présente sur la figure est issue de la définition du méta-modèle qui définit ainsi la structure. Nous pouvons donc y voir une décomposition de sections, puis de paragraphes et ensuite d'éléments unitaires.

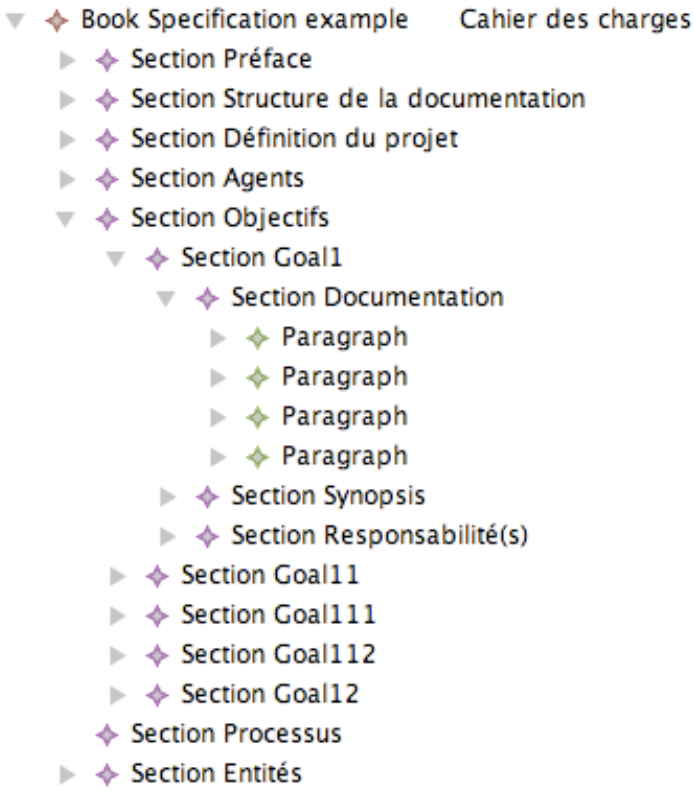


Figure 0-18 : Modèle de documentation

3. Génération

Plusieurs formats de génération s'offrent à nous : *DocBook*, *OpenDocument*, *RTF*[26]… Nous cherchons donc un format de fichier reconnu, éprouvé, facile à utiliser et générer. *RTF* ne présente donc pas toutes les caractéristiques nécessaires telle que la facilité de génération. En effet, les autres formats sont à base de XML donc beaucoup plus simples. *DocBook* est un format spécialisé et peu répandu. Il n'existe que très peu d'éditeur utilisant nativement *DocBook* et ceux-ci ne sont généralement peu distribués. Nous avons donc orienté notre choix vers le format *OpenDocument*. Celui-ci est un format de données ouvert pour les applications bureautiques. *OpenDocument* est la désignation d'usage d'une norme dont l'appellation officielle est OASIS Open Document Format for Office Applications, également abrégée par le sigle ODF. *OpenDocument* est avant

[26] *RTF* (Rich Text Format) est un format de fichier développé par la société Microsoft. Ce format descriptif non compressé est reconnu par la plupart des logiciels de traitement de texte. Sa vocation initiale est d'être un format pivot entre logiciels et plates-formes hétérogènes. Source : *wikipedia.org*.

tout proposé comme un format de convergence et d'interopérabilité entre logiciels bureautiques et, plus généralement, entre applications de traitement de documents. Un format *OpenDocument* est une archive compressée contenant un certain nombre de fichiers et de répertoires suivant une arborescence définie par la norme. Le seul fichier qui nous importe dans cette génération est le fichier *content.xml* contenant le contenu réel du document dont la syntaxe est inspirée du langage HTML.

La partie la plus intéressante du template de génération est celle concernant la génération des paragraphes (cf. Listing 5). Le script permet de parcourir l'ensemble des valeurs contenues dans le paragraphe et adapte son comportement en fonction du type de l'élément. Par exemple, nous générons un bloc de texte en utilisant la balise `<text:p>` alors que nous utiliserons les balises `<text:list>` et `<text:list-item>` pour les listes d'items.

```
<%script type="Documentation.Paragraph" name="content_paragraph"%>
<text:p text:style-name="Text_20_body">
<%for (values){%>
	<%if (filter("TextualValue") != null){%>
		<%filter("TextualValue").value%>
	<%}%>
	<%if (filter("EmphasisValue") != null){%>
		<text:span text:style-name="T1">
			<%filter("EmphasisValue").value%>
		</text:span>
	<%}%>
	<%if (filter("ItemizedListValue") != null){%>
		</text:p>
		<text:list text:style-name="L1">
			<%for (filter("ItemizedListValue").items){%>
			<text:list-item>
				<%content_paragraph%>
			</text:list-item>
			<%}%>
		</text:list>
		<text:p text:style-name="Text_20_body">
	<%}%>
```

```
        <%if (filter("XRefValue") != null){%>
                <text:a xlink:type="simple"
xlink:href="#<%filter("XRefValue").linkend.title%>"><%filter("XRefValue").li
nkend.title%></text:a>
        <%}%>
        <%if (filter("InformalTableValue") != null){%>
                </text:p>
                <%filter("InformalTableValue").content_table%>
                <text:p text:style-name="Text_20_body">
        <%}%>
<%}%>
</text:p>
```

Listing 5 : Template de génération à base d'un modèle de documentation

À l'issu de la génération, nous obtenons un fichier XML contenant l'ensemble du document auquel nous adjoignons plusieurs fichiers génériques tels que la feuille de style, les métadonnées du document, etc. pour enfin compresser ceux-ci et générer un fichier conforme au format *OpenDocument* utilisable avec la majorité des éditeurs de texte « grand public».

F. Synthèse

Nous présentons dans ce chapitre les transformations réalisées pendant ce travail de thèse. Nous avons choisi celles-ci car ce sont celles que nous avons eu besoin pour mener à bien nos cas d'étude. Il est important de noter qu'elles ne seront pas obligatoirement totalement adaptées à un autre contexte. La méthodologie SUN propose d'utiliser des interprétations pour améliorer la communication entre les différents participants mais ne spécifie pas lesquelles. L'interprétation concernant le prototype est toutefois relativement « universelle » et applicable à un grand nombre de contexte.

Le mécanisme d'interprétation proposé par la méthodologie SUN nous impose certaines contraintes telles que la définition de la transformation. Celle-ci doit prendre en entrée un modèle conforme à notre méta-modèle et définir un seul modèle en sortie. Cette contrainte impose donc, généralement, des adaptations au niveau du méta-modèle de sortie. Cela a été le cas pour l'interprétation vers

les cartes heuristiques. Nous avons du définir l'objet *Maps* afin de stocker plusieurs cartes heuristiques dans un seul modèle.

Une autre contrainte est définie par le mécanisme d'interprétation en imposant l'utilisant d'une unique transformation de modèles. Cela implique d'ajouter de l'intelligence au niveau de l'étape de génération. Comme on peut le voir sur l'interprétation de calcul du risque, nous transformons un modèle non technique dans un fichier JSON donc devant respecter une certaine syntaxe. Il aurait été possible d'utiliser un méta-modèle JSON et réaliser une transformation entre une spécification du besoin conforme à notre langage vers un modèle conforme au méta-modèle JSON. L'étape de génération aurait donc été grandement simplifiée. Toutefois, l'écart sémantique aurait été d'autant plus important entre les deux méta-modèles. La solution idéale aurait été une seconde étape de transformation d'un modèle conforme au méta-modèle d'analyse de risque vers un modèle JSON. Cette solution n'était pas envisageable dans ce contexte en raison de la définition de la structure d'une interprétation proposée par la méthodologie SUN. Nous avons du mettre de l'intelligence au niveau de la transformation mais également de la génération.

Application à la conception d'un outil de gestion de conférence

Le chapitre suivant a pour objectif de vous présenter une approche de conception logicielle guidée totalement par les modèles. Nous avons utilisé la méthodologie SUN pour la phase d'expression et de validation des besoins fonctionnels. La suite du processus est réalisée à l'aide des outils développés par la société BlueXML, partenaire de cette thèse.

BlueXML propose un environnement de développement permettant la génération de solutions logicielles à base de modèles. Le produit s'est spécialisé sur la gestion documentaire, et plus particulièrement sur la solution Alfresco[27]. A partir d'un ensemble de modèles inter connectés, il est possible de générer l'ensemble des fichiers de configuration nécessaires aux plate formes cibles. Six types de modélisation sont disponibles dans cet outil :

- Modélisation des données : Elle fait référence à un modèle de classe simplifié permettant de configurer les systèmes de stockage sous-jacents des plate formes.
- Modélisation des formulaires : Elle permet de définir des formulaires directement liés au modèle de données. L'ensemble des connecteurs entre les formulaires et les solutions de stockages sont mis à disposition.
- Modélisation des vues : Elle permet de définir un ensemble de vues connectées au modèle de données permettant de générer les composants graphiques mais aussi les sources d'accès au système de stockage.
- Modélisation des processus : Elle permet de spécifier des processus métier. Celle-ci est fortement inspirée du langage jPDL[28]. Il est également

---

[27] Alfresco offre une vraie alternative Open Source pour la Gestion de Contenu d'Entreprise (ECM) - Gestion de Document, Collaboration, Gestion des Archives/Enregistrements légaux, Gestion de Contenu Web et Gestion des Documents Numérisés. Alfresco dispose en tant que solution de gestion documentaire, d'une base de connaissances, de fonctions de gestion du cycle de vie du document, de travail collaboratif, de gestion des images. Il peut intégrer des contraintes réglementaires métiers en matière de classification, sauvegarde et conservation des informations.

[28] jPDL (jBPM Process Definition Language) est le langage de spécification de processus proposé par jBPM, moteur de workflow. Celui-ci permet la gestion de flux d'informations ainsi que la coordination entre biens et personnes.

possible de lier directement les étapes du processus à des formulaires pour indiquer les informations consultables et modifiables. Cette modélisation n'est pas illustrée dans ce cas d'étude malgré sa potentielle utilité. Il aurait été possible de définir un processus de soumission avec les liens de précédence entre objectifs. Cette possibilité n'a pas été exploitée car nous souhaitons nous comparer à des outils existants donc sans moteur de processus.

- Modélisation des portails : Il est possible de définir à travers l'outil une organisation de portail en décrivant chaque source de contenu. Cette modélisation n'est pas utilisée dans ce cas d'étude car aucun besoin ne s'est fait ressentir.
- Modélisation des besoins : Elle est directement issue des travaux de recherche réalisés pendant cette thèse.

Pour illustrer un processus de conception totalement orienté modèle, nous avons fait le choix de développer un outil de gestion de conférences. Celui-ci présente toutes les caractéristiques nécessaires pour être en adéquation avec notre processus de conception orienté modèle. C'est un système de consommation/production d'information où la nature des données varie en fonction de l'utilisateur et de l'activité. De plus, il a pour objectif de gérer des documents, c'est-à-dire des papiers scientifiques.

Afin de définir les besoins fonctionnels d'un tel outil, nous nous sommes fortement inspirés de l'outil OpenConf[29]. C'est un outil web permettant de disposer d'un site de soumission avec un processus de relecture relativement simple. Celui-ci est aujourd'hui très utilisé dans le domaine scientifique.

A. Modèle de besoin

Afin de définir notre modèle de besoins, nous avons parcouru le site de démonstration mis à disposition puis nous avons isolé l'ensemble des besoins fonctionnels et synthétisé le modèle de besoin présent sur la Figure 0-1. Sur cette

[29] http://www.openconf.com/

figure, nous avons colorisés les objectifs en fonction de leur profondeur afin de simplifier la lecture. L'objectif principal se nomme « *Manage conference* ». Celui-ci représente l'objectif stratégique de l'application. Il se décompose en quatre sous-objectifs :

- « *Create submission* » pour le dépôt de soumission ;
- « *Manage submissions to review* » pour la relecture des soumissions ;
- « *Manage submissions to advocate* » pour la décision finale ;
- « *Organize conference* » pour obtenir les informations nécessaires à l'organisation.

Chacun de ces objectifs se décomposent ensuite pour affiner le besoin réel. Le dépôt de soumission se décomposera par la création d'une soumission, l'édition d'une soumission et enfin l'annulation d'une soumission. Il en sera de même pour les autres objectifs.

Pour simplifier la lecture du modèle de besoins, nous nous concentrons ensuite sur l'agent représentant un auteur. Nous analysons seulement les objectifs sous sa responsabilité et leurs politiques d'accès associées.

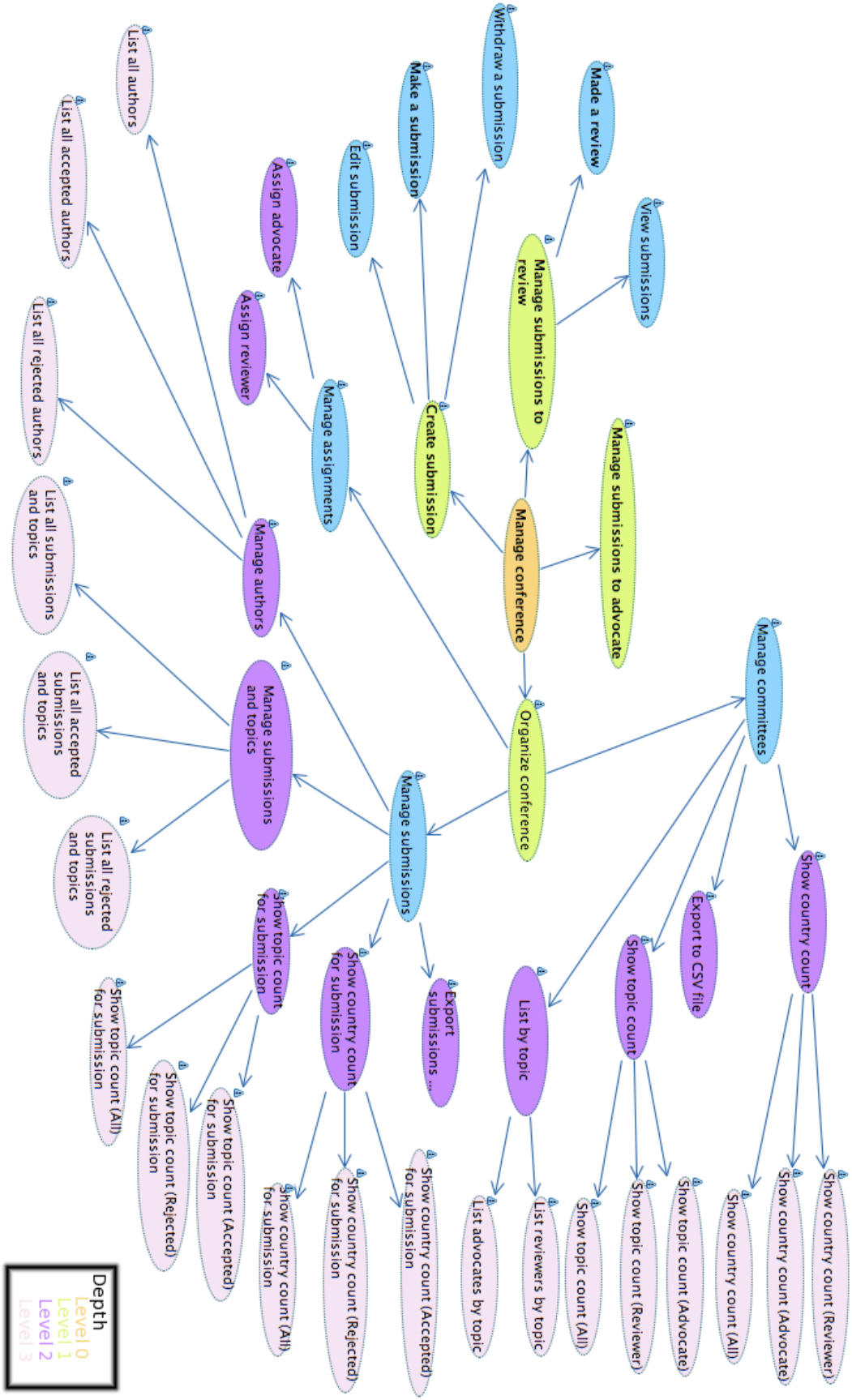


Figure 0-1 : Modèle de besoin correspondant à l'outil de gestion de conférence

Après avoir défini l'ensemble des objectifs de l'outil, il s'agit ensuite de définir les agents ainsi que leurs responsabilités (cf. Figure 0-2). Nous définissons dans notre spécification quatre agents :

- « *Author* » chargé de soumettre des papiers et donc responsable de l'objectif « *Create submission* » ;
- « *Reviewer* » chargé de relire les soumissions et donc responsable de l'objectif « *Manage submissions to review* » ;
- « *Advocate* » chargé de prendre la décision finale au sujet des soumissions et donc responsable de l'objectif « *Manage submissions to advocate* » ;
- « *Program Chair* » chargé d'organiser la conférence donc responsable de l'objectif « *Organize conférence* » mais inclus également les responsabilités d'un « *Advocate* ».

Figure 0-2 : Definition des agents et de leurs responsabilités

Pour un souci de lisibilité, nous ne faisons figurer sur la Figure 0-2 que les objectifs sous la responsabilité directe d'un agent. Il faut tout de même extraire de cette figure qu'un agent est en charge des objectifs sous sa responsabilité mais également de leurs sous-objectifs. Un auteur est donc responsable également des objectifs suivants : « *Withdraw a submission* », « *Make a submission* » et « *Edit a submission* ».

Afin de vérifier la bonne répartition des activités entre les différents agents, il est possible d'utiliser l'interprétation vers les cartes heuristiques permettant d'isoler les objectifs pour chacun des agents (cf. B Approche par les cartes heuristiques ou « mind maps »). On peut voir sur la Figure 0-3 la décomposition des objectifs sous la responsabilité d'un auteur.

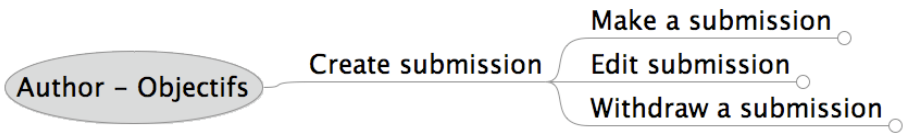


Figure 0-3 : Carte heuristique représentant la liste des objectifs d'un auteur

Après avoir défini les objectifs, les agents ainsi que leurs responsabilités, il s'agit de définir le dictionnaire de données (cf. Figure 0-4). Celui-ci est relativement simple car il ne pas fait l'objet de beaucoup de questionnement. Le principal concept est celui de soumission (*Paper*) caractérisé par son titre, des mots-clefs, un commentaire et un résumé. Celui ci est relié à des auteurs (*Author*) et à des thèmes (*Topic*). La notion de revue (*Review*) est également un concept important à lier à une soumission. Elle permet d'apporter un avis sur une soumission déposée par un auteur.

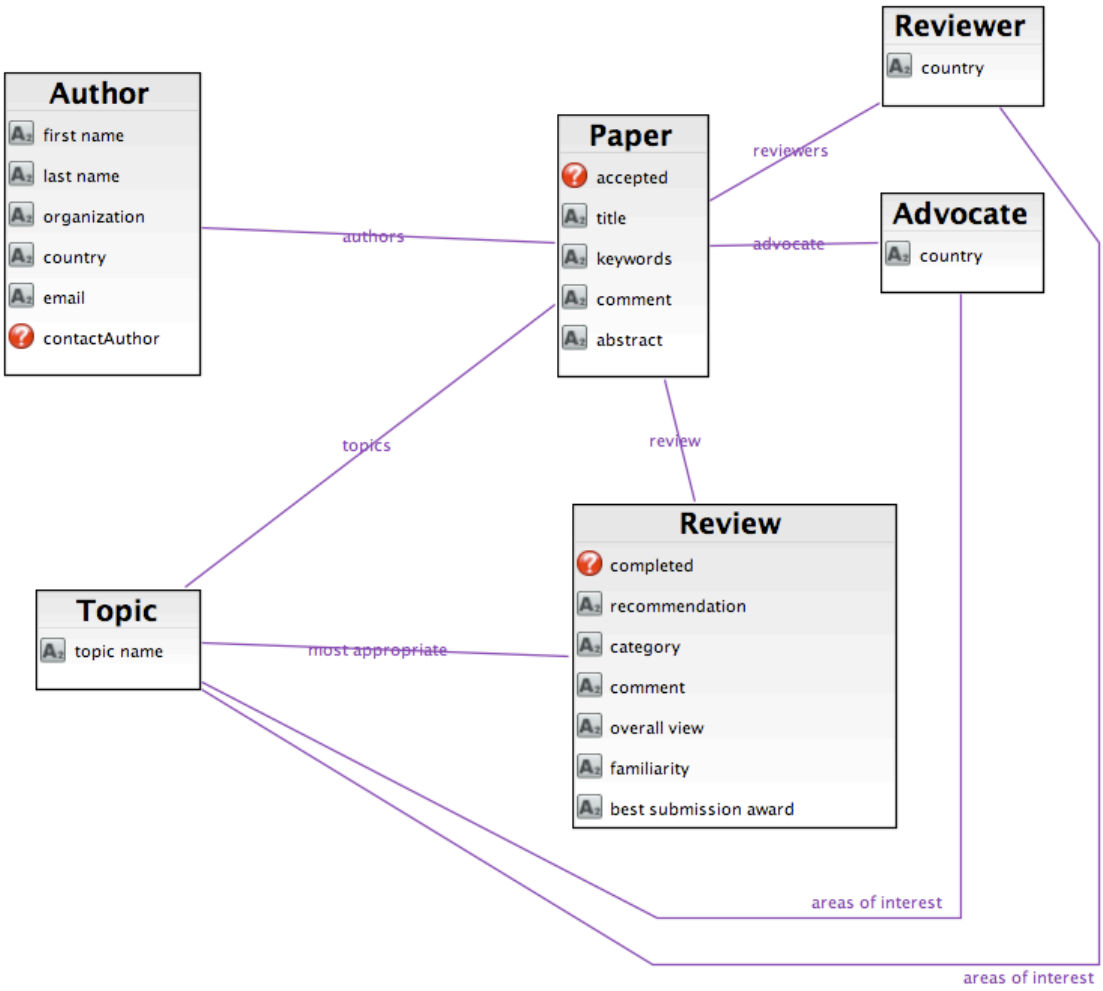


Figure 0-4 : Dictionnaire de données du système de gestion de conférence

Après avoir défini les objectifs ainsi que le dictionnaire de données, nous pouvons procéder à la définition des politiques d'accès (cf. Figure 0-5). Nous pouvons voir que les trois objectifs incluant une politique d'accès ont comme point d'entrée le concept de soumission (*Paper*). Afin de créer une soumission (*Make a submission*), l'auteur doit pouvoir créer une soumission, lire les informations nécessaires ainsi que mettre à jour les champs. Ces droits sont synthétisés dans les trois lettres *CRU* à côté de *Paper*. L'auteur peut ensuite mettre à jour les champs suivants : titre, mots clefs, commentaire et résumé. Il ne peut par contre pas accéder au champ indiquant l'acceptation de la soumission. On peut également voir qu'il peut lire et modifier les associations *authors* et *topics*. Les politiques d'accès liées à ces deux concepts sont différentes entre ces deux associations. En effet, l'auteur ne peut créer de nouveaux thèmes (*Topic*), il devra sélectionner un thème qui a été préalablement créé. Par contre, il peut créer et supprimer des auteurs. Le formulaire ne devra pas proposer les mêmes composants graphiques pour ces deux associations.

Pour l'édition d'une soumission (*Edit submission*), la politique d'accès est semblable à celle définie pour la création d'une soumission. La seule différence est que le droit accordé pour une soumission n'est plus *CRU (Create, Read, Update)* mais *RU (Read, Update)*, ce qui indique qu'il ne sera pas possible de créer une nouvelle soumission pendant l'activité supportée par cet objectif.

Pour la suppression d'une soumission (*Withdraw a submission*), l'auteur peut voir les soumissions ainsi que leur titre mais peut aussi les supprimer. Ces droits sont synthétisés dans *RD (Read, Delete)*.

Nous définissons ensuite une politique d'accès pour chacun des objectifs devant supporter une activité utilisateur.

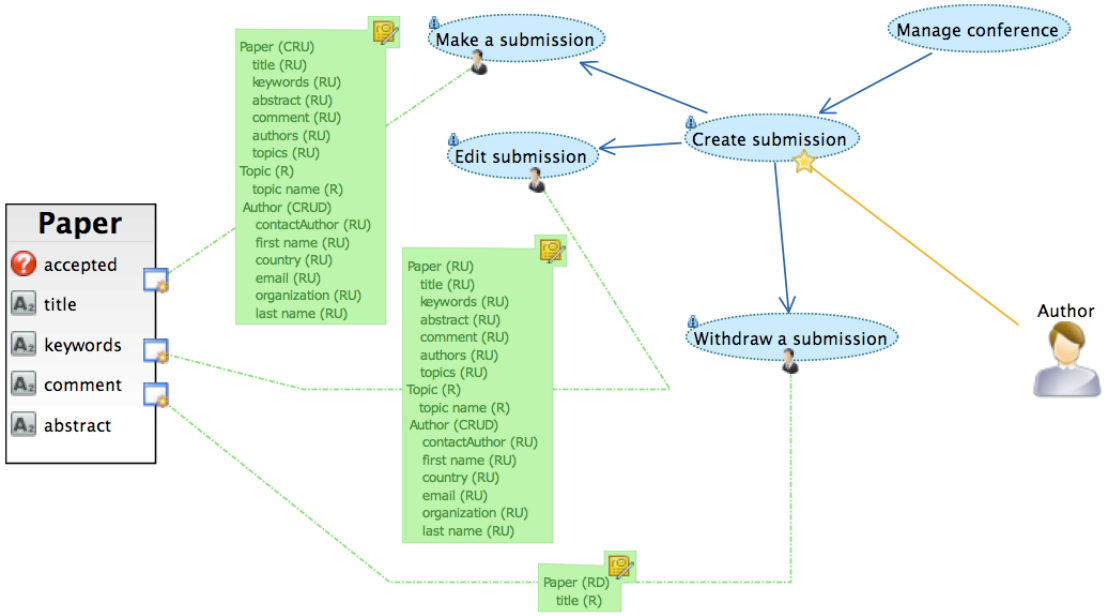


Figure 0-5 : Politiques d'accès définies pour les objectifs sous la responsabilité d'un auteur

Afin de vérifier que chacun des agents n'accède pas à des informations auxquelles il ne devrait pas accéder, nous pouvons également utiliser les interprétations vers les cartes heuristiques. Il est possible d'utiliser l'interprétation qui fusionne l'ensemble des politiques d'accès afin de vérifier la vision du système d'information par chaque agent. Pour l'auteur (cf. Figure 0-6), on peut ainsi voir qu'il n'a pas accès au champ *accepted* d'une soumission. Il n'a pas non plus accès aux relectures faites. Nous supposons dans cet outil qu'elles sont envoyées par mail. Un autre objectif aurait pu être défini afin d'y accéder en lecture.

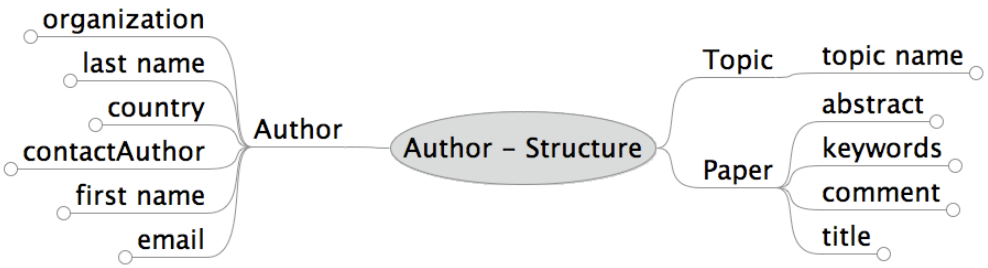


Figure 0-6 : Carte heuristique synthétisant la vue du dictionnaire de données par un auteur

B. Analyse des risques

L'analyse des risques va nous permettre d'identifier les éléments sur lesquels l'équipe de conception devra se concentrer. Les formules utilisées sont celles présentées dans la partie *C* Évaluation des risques.

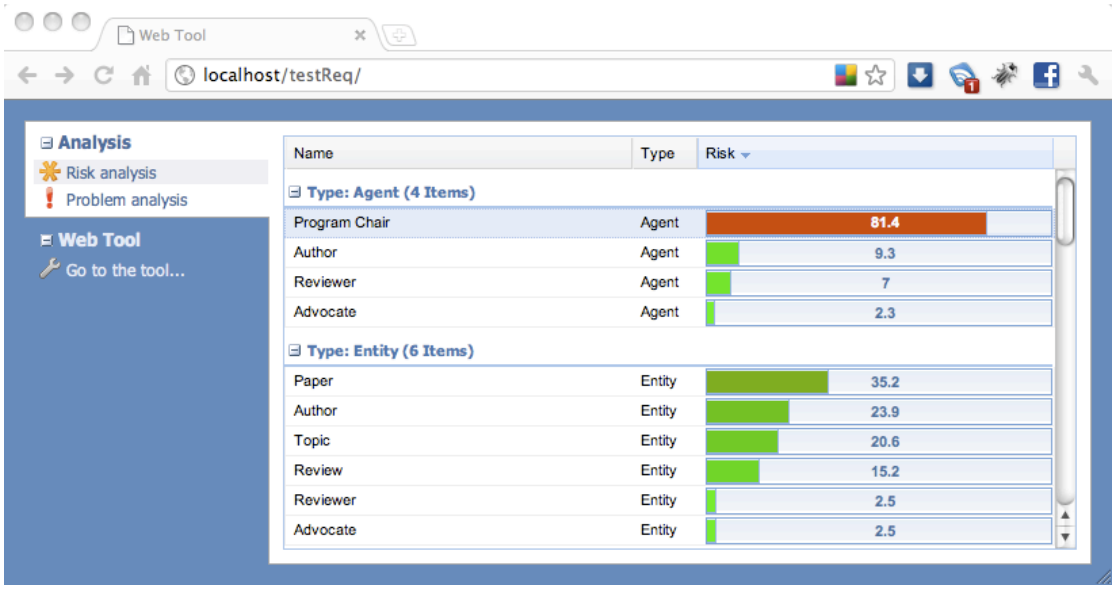


Figure 0-7 : Analyse des risques sur les agents et les entités

La Figure 0-7 nous montre les risques calculés pour les agents ainsi que les entités. Avec une valeur de 81.4, l'agent « *Program Chair* » doit être l'acteur sur lequel la majorité des efforts doit se concentrer. Celui-ci obtient un tel score en raison du nombre important d'objectifs sous sa responsabilité. En ce qui concerne le dictionnaire de données, il est important de vérifier que les définitions d'une soumission (*Paper* avec un score de 35.2) et d'un auteur (*Author* avec un score de 23.9) sont correctement définis et acceptés par l'ensemble des participants. Ces concepts sont mis en avant car fréquemment utilisés dans les différentes politiques d'accès. Il est en effet logique que la notion de soumission soit réutilisée dans la majorité de l'outil.

La Figure 0-8 permet de visualiser les valeurs de risque pour les objectifs. Étant donné que nous n'avons pas spécifié de priorité sur les objectifs, le calcul du risque n'est influencé que par la dimension de la politique d'accès et le nombre d'agents. L'objectif à plus haut risque est l'objectif suivant « *Manage submissions to advocate* ». Cela s'explique par la dimension de la politique d'accès. En effet,

afin de remplir cette tâche, l'acteur responsable doit accéder à la soumission, ses auteurs liés ainsi que ses thèmes et les relectures faites. Le deuxième objectif à plus haut risque est « *Made a review* ». Celui-ci a une politique d'accès similaire au précédent à la seule différence qu'il n'est pas possible d'accéder aux auteurs de la soumission ce qui fait que son risque est plus faible. Les objectifs à haut risque sont à bien analyser car ils représentent des fonctionnalités pour lesquelles il sera potentiellement difficile d'apporter une réponse adaptée. Dans notre cas, le risque est présent car les fonctionnalités précédentes doivent permettre de consulter et de modifier une quantité importante d'information. Les interfaces proposées seront donc plus difficiles à réaliser afin d'atteindre un niveau d'utilisabilité et de clarté satisfaisant.

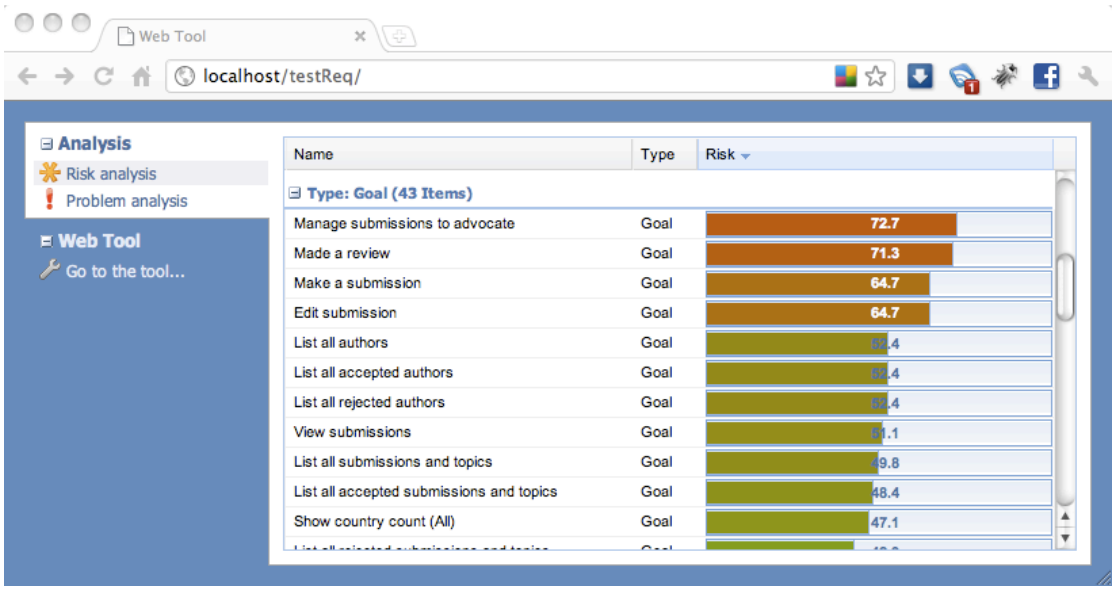


Figure 0-8 : Analyse des risques sur les objectifs

C. Prototypage

Après avoir spécifié le besoin fonctionnel et analysé le risque de chacun des éléments, il est possible de faire intervenir les utilisateurs finaux à travers le prototype. Il est ainsi possible de tester directement les objectifs et leurs politiques d'accès associées.

Pour chacun des objectifs, il est possible de tester et d'annoter directement le prototype (cf. Figure 0-9). L'ensemble des annotations est ensuite réintégré à la

spécification du besoin. Chacune d'elle devra être traitée et une réponse devra être apportée par l'individu chargé de l'évolution de la spécification du besoin.

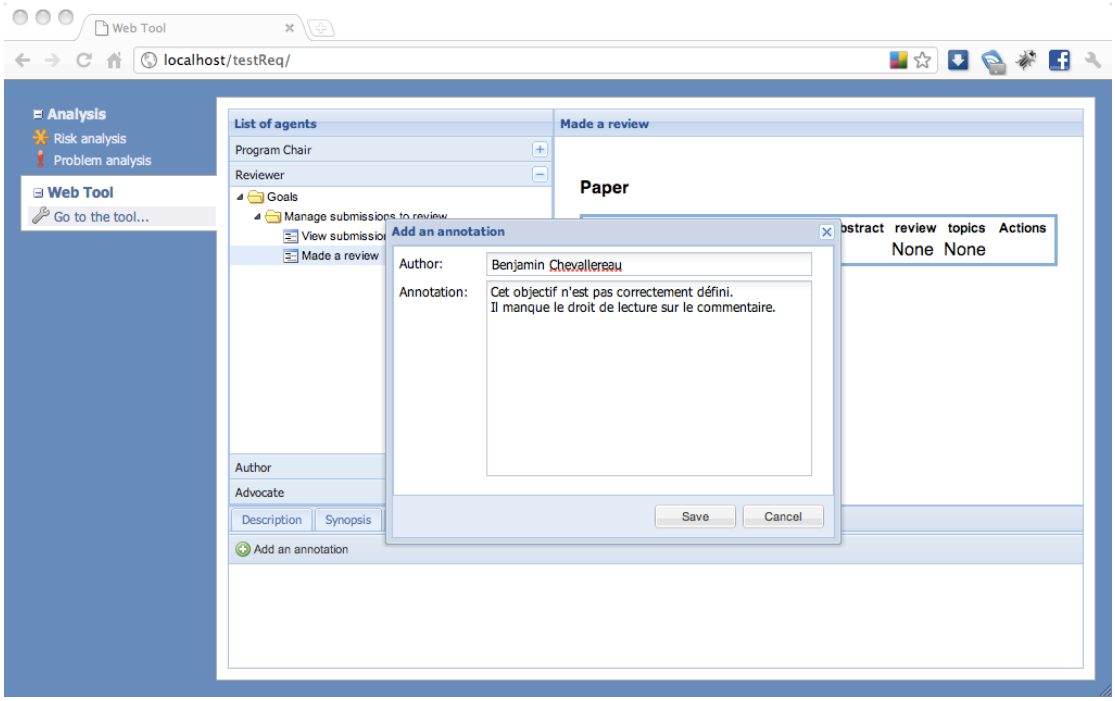


Figure 0-9 : Prototype web obtenu à partir de la spécification du besoin

D. Génération des modèles de conception

A l'issu de la spécification du besoin, il est possible d'obtenir directement les modèles de conception utilisables par les outils développés par la société BlueXML.

1. Modèle de donnée

Le premier modèle obtenu est le modèle de données. Celui-ci peut être assimilé à un diagramme de classes UML (cf. Figure 0-10). Nous retrouvons sur cette figure l'ensemble des entités, attributs et relations définis dans le dictionnaire de données de la spécification du besoin. On peut noter que les attributs de type indéfini dans la spécification du besoin sont transformés en attribut de type chaine de caractères comme *accepted* dans la classe *Paper*. Les types adéquats devront donc être redéfinis.

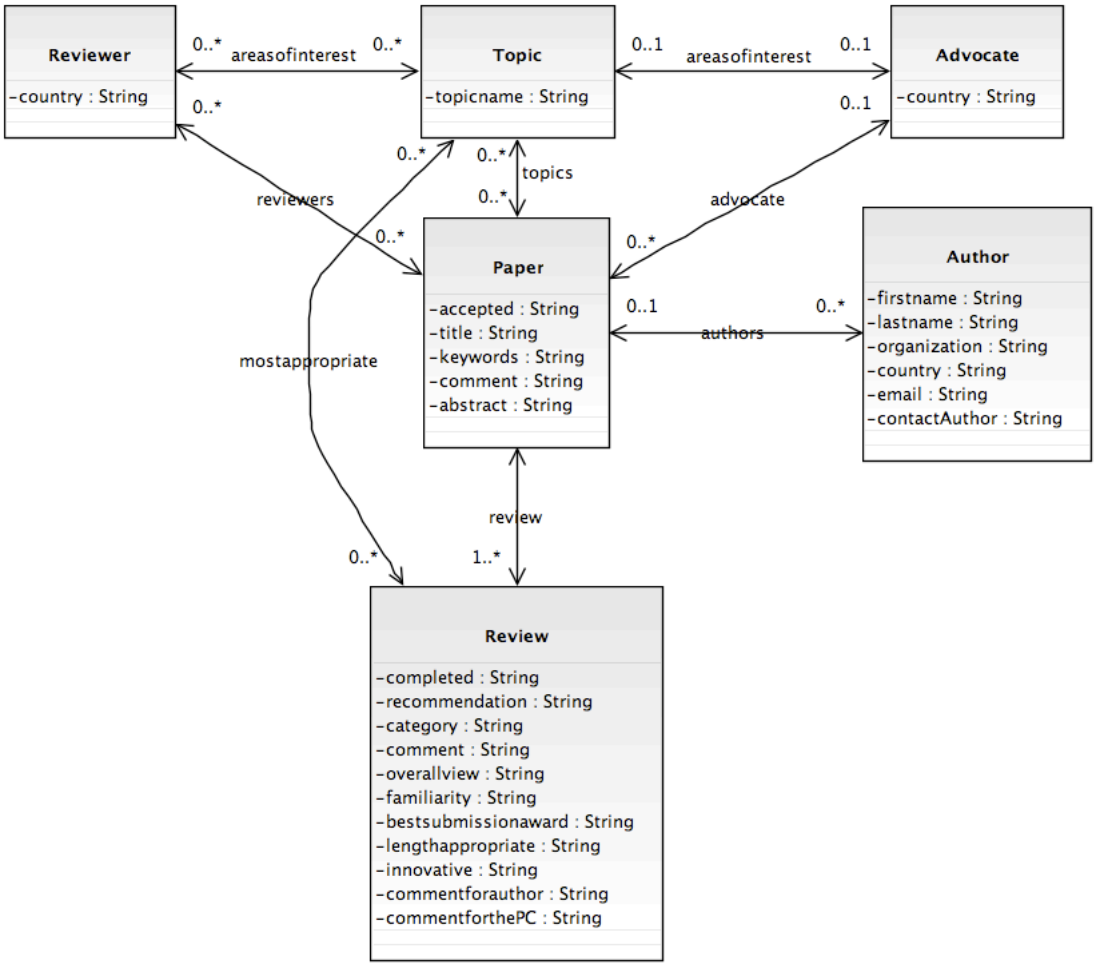


Figure 0-10 : Modèle de données obtenu après transformation

D'autres modifications sont à effectuer sur le modèle présenté ci-dessus. Alors que la notion d'auteur est une information à stocker, les notions de relecteur (*Reviewer*) et juge (*Advocate*) ne sont pas nécessaires. En effet, celles-ci représentent des profils d'utilisateur dans le futur outil. Les associations liées à ces concepts sont donc supprimées de la même manière.

Différentes énumérations, ou liste de valeurs, doivent également être crées et reliés à des attributs. Par exemple, le champ *category* de la classe *Review* sera lié à l'énumération *Category* qui contient comme valeur : « *Highly theoretical* », « *Tends towards theoretical* », « *Balanced theory and practice* », « *Tends toward practical* », « *Highly practical* ». Le modèle obtenu est présenté sur la Figure 0-11.

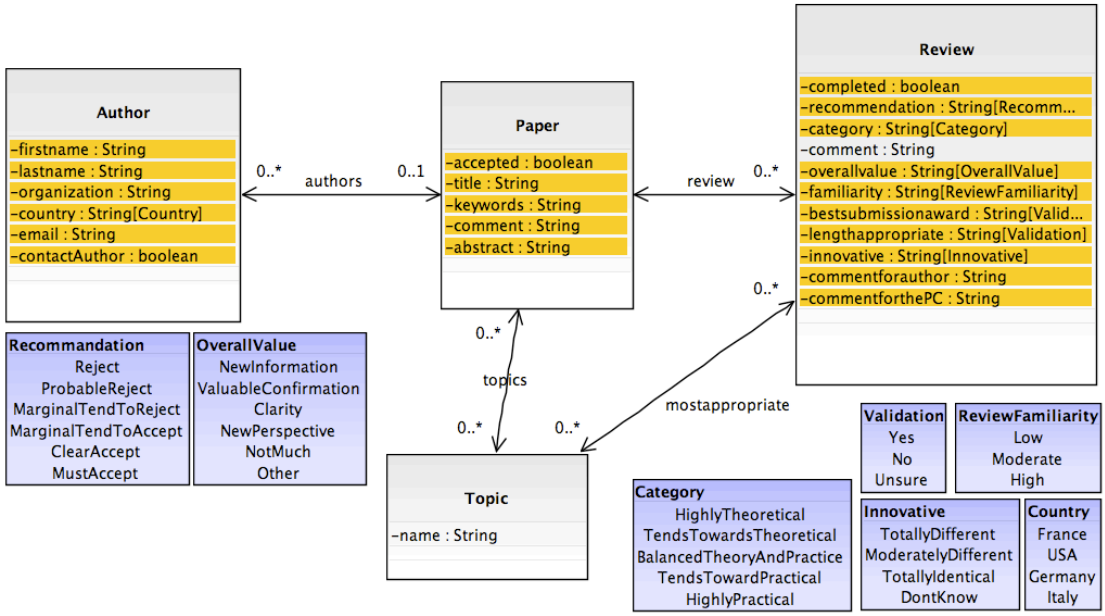


Figure 0-11 : Modèle de données obtenu après raffinement

2. Modèle de formulaire

Le modèle de formulaire obtenu contient un formulaire pour chaque objectif incluant une politique d'accès. Dans certains cas, cela n'a pas de sens car aucune modification des données n'est possible. C'est le cas par exemple pour l'objectif « *List all authors* ». La fonctionnalité issue de cet objectif ne fera appel à aucun formulaire. Toutefois, il est possible d'utiliser un modèle de vue. Le modèle de formulaires est donc amputé d'un bon nombre de formulaires. Dans notre cas, nous passons de trente six formulaires à cinq formulaires. Pour les objectifs non couverts par un formulaire, nous définissons des vues. Celles-ci permettent par la suite d'accéder directement à la source de données en toute simplicité.

E. Génération et raffinage sur une solution de gestion documentaire

A l'issu de la génération, la plateforme de stockage est quasiment généré. Quelques modifications sont nécessaires afin de pouvoir appliquer l'ensemble des filtres nécessaires à la sélection des données. La majeure partie du travail consiste à réaliser la partie visuelle de l'application (cf. Figure 0-12). Pour

réaliser celle-ci, nous allons utiliser une librairie qui s'appelle *ExtJS*[30]. C'est une bibliothèque permettant de construire des applications web interactives.

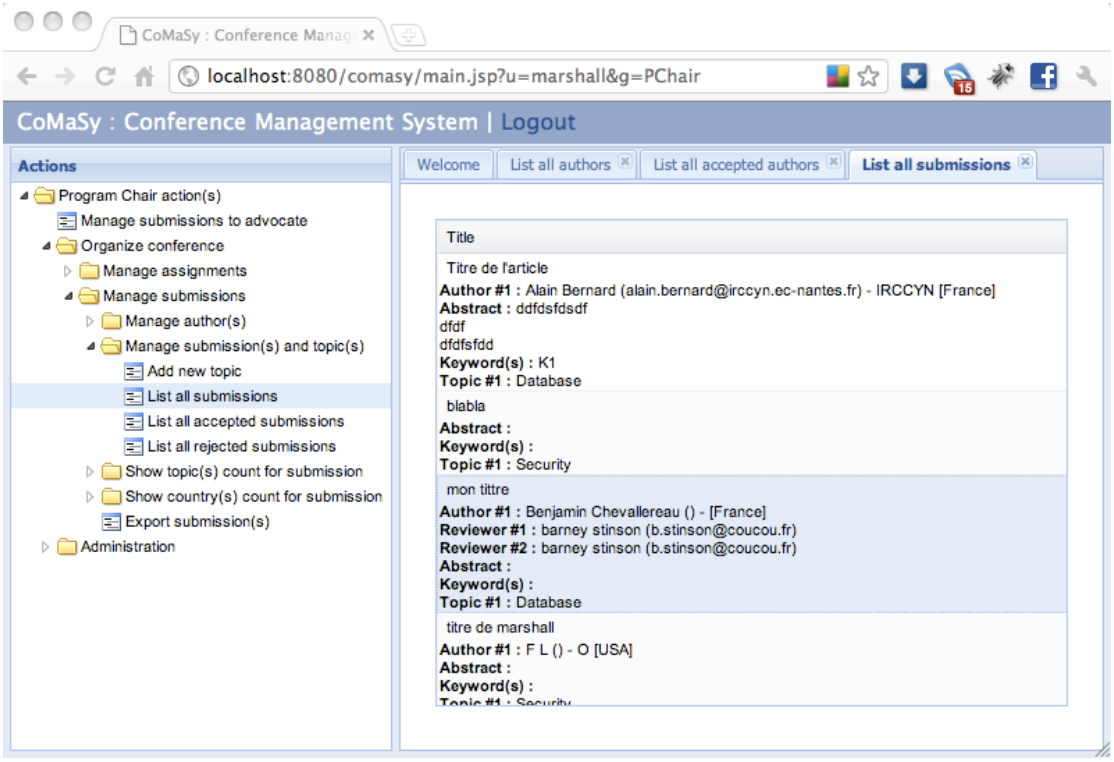


Figure 0-12 : Outil développé de gestion de conférences

F. Synthèse

Ce cas d'étude démontre comment utiliser la méthode SUN et, de manière plus large, comment instrumentaliser complètement un processus de conception orienté modèle. Celui-ci a été réalisé sur un système de gestion de conférences.

Le premier axe d'analyse peut être la quantité d'informations définies et déduites de la précédente étape. Pour cela, nous allons utiliser comme mesure l'octet, et plus particulièrement le kilo-octet (ko), pour mesurer la quantité de données. A l'issu de la phase de définition du besoin, notre spécification véhicule 43ko de données. A partir de celle-ci, nous déduisons les modèles de conception avec un volume de 44ko de données, soit quasiment le même volume. A l'issu de l'étape de raffinement des modèles, les modèles de conception ne représentent

[30] http://www.sencha.com/products/js/

plus que 25ko en raison des nombreux éléments inutiles donc supprimés. L'étape de génération nous permet d'obtenir 281ko de données à partir des modèles de conception. Le gain s'explique par les nombreux fichiers générés et de leurs verbosités. L'étape de génération nous fournit des fichiers bruts qu'il faut ensuite encapsuler dans des composants génériques préalablement développés. A l'issu de cette étape, nous obtenons 17 692ko de données. Toutefois, cette valeur n'est pas très significative en raison de la quantité importante de fonctionnalités proposées par ce composant mais non utilisées dans notre cas d'étude. Le processus de développement s'est ensuite concentré sur deux parties. La première concerne les améliorations faites sur la plate-forme de stockage qui représente seulement 25ko de données en comparaison au 281ko de données générées. La seconde partie concerne le développement de l'outil web. Le fruit de ce travail a permis d'obtenir 77ko de données qui ont ensuite été intégrées avec une bibliothèque.

Ce qu'il faut retenir de tous ces chiffres est que nous perdons de l'information entre la spécification du besoin et les modèles de conception finaux. Cela s'explique par le raffinement réalisé et la centralisation des informations nécessaires. Par exemple, les trois objectifs chargés de lister les auteurs ne sont représentés dans les modèles de conception que par une seule vue sur la classe correspondante à l'auteur. Il est également intéressant de voir que seulement 8% de la configuration de la plate-forme de stockage a été réalisée manuellement. Tout le reste a été généré à partir des modèles de conception.

Le second axe d'analyse peut être le temps passé sur chaque étape du processus. On peut comparer les temps mesurés (cf. Tableau 0-2) avec les valeurs théoriques proposées par la méthode de répartition proportionnelle [MOR 06] (cf. Tableau 0-1). Étant donné que nous étions dans le contexte d'un cas d'étude, certaines étapes n'ont pu être mesurées comme l'encadrement de projet, la recette, etc. Pour calculer les répartitions de charges manquantes dans notre cas d'étude, nous avons utilisé les pourcentages théoriques (cf. lignes grisées sur le Tableau 0-2). Nous pouvons voir que 40% de la charge du projet est affectée à l'étude préalable, ce qui couvre la spécification du besoin, soit quatre fois plus que le pourcentage théorique. Cela permet de montrer le basculement des efforts

vers les premières étapes du projet. En raison de l'automatisation et de la validation par prototypage, l'étude détaillée est fortement amputée. La charge de réalisation est, dans notre processus, équivalente à la charge de l'étude préalable. Ces chiffres doivent être utilisés avec précaution car obtenus à partir d'un seul projet. Toutefois, il est important de noter une réelle différence de répartition des charges avec un processus de conception orienté modèle. La définition du besoin est approfondie et validée dans les premières étapes du projet.

Étape	Ratio
Étude préalable	10% du total du projet
Étude détaillée	25% du total du projet
Étude technique	10% de la charge de réalisation
Réalisation	Deux fois la charge d'étude détaillée
Encadrement du projet - à l'étape de réalisation, - aux autres étapes.	 20% de la charge de réalisation 10% de la charge de l'étape
Recette	20% de la charge de réalisation
Documentation	5% de la charge de réalisation

Tableau 0-1 : Répartition proportionnelle de la charge entre les étapes

Étape	Volume	Ratio
Étude préalable	30 heures	40% du total du projet
Étude détaillée	5 heures	5 % du total du projet
Étude technique	3 heures	10% de la charge de réalisation
Réalisation	30 heures	40% du total du projet
Encadrement du projet	2 heures	20% de la charge de réalisation 10% de la charge pour les autres étapes
Recette	6 heures	20% de la charge de réalisation
Documentation	1,5 heures	5% de la charge de réalisation

Tableau 0-2 : Répartition des charges effectives

Ce cas d'étude nous montre qu'il est possible de définir un processus complet de conception orienté modèle. Nous démontrons que la méthodologie SUN ainsi que les outils proposés par BlueXML permettent de faire un gain sur les phases de réalisation pour se concentrer sur les étapes de définition du besoin. Ce processus a été appliqué sur d'autres exemples que l'organisation de conférences. Nous avons fait par exemple un cas d'étude dans le domaine de la finance (cf. I.B.Annexe D) et un autre dans l'univers hospitalier (cf. I.B.Annexe C).

Ceux-ci ont montré de bons résultats mais aucune mesure du temps n'a été réalisé.

Conclusions et perspectives

Malgré les efforts accomplis par l'industrie logicielle pendant ces dernières années, elle n'arrive pas à répondre pleinement à l'ensemble des défis qui lui sont posés. Elle répond de plus en plus efficacement aux attentes technologiques mais plus difficilement aux attentes humaines. Elle rencontre des difficultés pour communiquer avec les experts métier concentrant les connaissances nécessaires à la bonne conception d'un SI. En effet, les concepteurs et les experts métier ne partagent pas le même langage. Ce problème entraine des incompréhensions impliquant parfois la conception d'outil inadapté à l'activité supportée.

L'objectif de ce travail de recherche est d'améliorer la phase de spécification du besoin. Celle-ci regroupe les activités de définition, communication et diffusion des besoins fonctionnels à l'ensemble des participants d'un projet.

Une possibilité est de proposer un langage commun de définition d'outils, compréhensible par l'ensemble des participants d'un projet de développement. Mais à ce jour, aucune proposition n'a réussi à satisfaire l'ensemble des acteurs.

De plus, le pourcentage de temps accordé à la phase de spécification du besoin est généralement très inférieur à celui concernant la phase de développement. L'hypothèse selon laquelle le temps nécessaire à l'étape de maintenance, et plus minoritairement, l'étape de développement, décroît en fonction du temps accordé à la phase de spécification du besoin est prouvé par de nombreuses recherches. Toutefois, il est impératif que le temps accordé à cette étape initiale ne soit pas trop important. Il est donc nécessaire de proposer des mécanismes canalisant les étapes trop consommatrices.

L'état de l'art nous montre que les cycles traditionnels de développement laissent peu de place à la phase de spécification du besoin. De plus, les experts métier sont généralement peu consultés lors des phases suivantes. Les méthodes agiles prennent le contre-pied de cette constatation en faisant intervenir à des niveaux différents les experts métier. Toutefois, la phase de spécification du besoin n'est pas plus approfondie dans ces méthodes.

Des méthodes existent pour analyser et définir plus précisément la spécification du besoin. Certaines reposent sur des techniques d'analyse, participative ou expérimentales. L'inconvénient majeur est la logistique nécessaire au bon déroulement de ces méthodes.

Différentes approches proposent également des langages d'expression du besoin. Ceux-ci sont catégorisés dans les langages à base de scénario ou orienté autour de la notion d'objectif. Généralement complexe, généraliste et difficile à mettre en œuvre, ceux-ci ne sont généralement pas exploités dans l'industrie. Cela s'explique par des langages trop riches et génériques.

Un langage de modélisation a été proposé dans ce manuscrit. Celui-ci est basé sur la notion d'objectif et fait également appel aux notions proposées par le modèle ERA pour définir le dictionnaire de données. A ces deux grandes parties du langage, un concept « glue » a été proposé : une politique d'accès permettant de définir la vue du dictionnaire de données pour chacun des objectifs. Ce langage est spécialisé pour la conception de SII et a pour objectif d'être le plus simple possible afin de faciliter sa prise en main.

La résistance au changement est un facteur très important dans l'acceptabilité d'une nouvelle technologie, méthode ou fonctionnalité. Afin de réduire celle-ci, nous proposons le mécanisme d'interprétation. Il doit permettre de simplifier l'accès à la spécification du besoin pour les participants les plus réfractaires. Ce mécanisme cherche à promouvoir le fait que cette méthodologie doit s'adapter aux acteurs et non l'inverse. Il est également utile dans le cas d'un effort de synthèse qui peut être automatisé et ainsi apporter un gain en terme de productivité.

Le mécanisme d'interprétation a été réalisé en utilisant les artefacts proposés par l'ingénierie dirigée par les modèles dans un effort de gain de productivité. Différentes interprétations ont été réalisées : cartes heuristiques, outils d'analyse et de calcul de risque, exemple de prototype web, etc.

A l'issu de l'étape de spécification du besoin, la phase de conception est initialisée à partir des résultats de l'étape précédente. Ce passage représente le

point de sortie de la méthodologie proposée dans ce manuscrit. Afin d'illustrer ce passage, nous avons créé des transformations afin de générer les modèles de conception adaptés à l'environnement de développement proposé par la société partenaire de cette thèse.

Cette méthodologie avec l'ensemble de ses composants, c'est-à-dire son langage, ses outils de définition ainsi que ses différentes interprétations, a été testée sur différents cas d'étude. Celui abordé dans le dernier chapitre, concernant le système de gestion de conférences, est un cas d'étude scientifique et a fait l'objet d'une étude approfondie. Toutefois, d'autres analyses ont été réalisés dans des contextes totalement différent tels que la diffusion d'information entre une organisation hospitalière et son tissu local, la gestion de documents dans un contexte de fusion/acquisition d'entreprise ou encore la dématérialisation de document [CHE 09b].

Ces travaux de recherche ont également été un laboratoire « grandeur nature » d'un cycle de développement logiciel totalement orienté modèle. Celui-ci utilise la méthodologie SUN proposée dans ce manuscrit, puis l'environnement de conception proposée par la société BlueXML en utilisant une transformation automatique. A l'issu de la conception, nous avons utilisé les mécanismes de génération proposées par le même environnement. Nous avons ainsi pu conclure qu'il était possible de définir un cycle de développement logiciel orienté modèle en faisant intervenir les experts métier dès les premières phases. De plus, nous avons estimé les gains de productivité acquis pendant ce cycle.

L'apport principal de ces travaux porte sur l'ingénierie des besoins et plus particulièrement l'ingénierie des besoins orientée autour du concept d'objectif. Ces travaux se placent également dans le champ proposé par l'ingénierie dirigée par les modèles sur laquelle nous avons structuré l'ensemble de notre proposition.

Ces travaux ouvrent des perspectives scientifiques à de futurs travaux de recherche sur la complémentarité de l'ingénierie des besoins et l'ingénierie dirigée par les modèles. Le langage proposé se concentre sur la définition de SII. Une adaptation de ce langage au domaine d'activité devrait être étudiée. De

nombreuses perspectives industrielles sont également envisageables. L'ensemble des outils proposés dans cette méthodologie est déjà intégré à l'environnement de développement de la société BlueXML. L'ensemble des interprétations proposé dans ces travaux n'est qu'une illustration du mécanisme. De très nombreuses autres interprétations peuvent être réalisées en fonction des finalités attendues.

Les différents cas d'étude, scientifiques et industriels, ont permis de valider la proposition tant au niveau langage, outil, interprétation qu'au niveau méthodologie. Nous faisons l'hypothèse dans ces travaux que notre langage est plus simple en terme de compréhension et rédaction. Toutefois, nous n'avons pas réalisé d'étude réelle. Nous avons étudié les travaux de S. Patig [PAT 08a, PAT 08b] et développé un outil pour réaliser cette expérimentation mais celle-ci n'a pas été menée à son terme en raison des difficultés rencontrées à regrouper un panel d'utilisateur représentatif de la population que l'on souhaitait tester.

Il n'existe pas aujourd'hui de point d'entrée simplifié dans la méthodologie. Le responsable métier doit définir à chaque projet un modèle de spécification du besoin pour lequel nous ne proposons pas réellement de mécanismes de réutilisabilité. Deux approches peuvent être explorées. La première concerne la définition de librairies de modèles permettant de synthétiser une définition commune et partagée d'un dictionnaire de données spécifiques à un domaine d'activité. Cette approche permettrait ainsi d'accélérer l'initialisation d'un modèle de besoin. La seconde approche consiste à proposer un mécanisme d'analyse textuelle permettant d'extraire les concepts principaux émergents du document. Avant un lancement de projet, un document synthétique est rédigé pour supporter les discussions. Il serait utile de proposer un mécanisme permettant d'initialiser la spécification du besoin avec cette catégorie de document.

Ces travaux ont permis de répondre à des attentes scientifiques en termes de langage de définition de SII adapté aux experts fonctionnels et d'une proposition de nouvelle méthodologie de conception. Nous avons industrialisé l'ensemble de notre proposition et nous avons démontré la possibilité de définir un cycle de

développement logiciel totalement dirigé par les modèles. De plus, celui-ci a montré dans des cas d'étude montrant de très bons résultats en terme de productivité.

Bibliographie

A. Valorisation des travaux de thèse

Publications dans des revues d'audience internationale à comité de lecture

[CHE 10c] Chevallereau, B., Chenouard, R., Bernard, A., Mévellec, P., 2010, *Model Driven Requirements Engineering (MDRE): MDE in the first step of the model driven development process*, Journal of Information Systems [En cours de soumission]

Communications à des congrès internationaux à comité de sélection et actes publiés

[CHE 09a] Chevallereau, B., Bernard, A., 2009, *Améliorer les performances de l'industrie logicielle par une meilleure compréhension des besoins*, 8ème Congrès International de Génie Industriel, 10-12 juin 2009, Bagnères de Bigorre, France.

[CHE 09b] Chevallereau, B., Bernard, A., 2009, *How to build web self-services by functional profiles?*, 3rd International Conference on New Technologies, Mobility and Security, 20-23 décembre 2009, Le Caire, Égypte.

Colloques sans actes ou avec actes à diffusion restreinte

[CHE 08a] Chevallereau, B., 2008, *Durabilité des applications : Contribution des nouvelles approches de modélisation*, Colloque de Recherche de l'Intergroupe des Écoles Centrales, 09-10 juin 2008, Nantes, France.

[CHE 08b] Chevallereau, B., 2008, *Améliorer les performances de l'industrie logicielle par une meilleure compréhension des besoins*, 11ème journées STP du GDR MACS, 20-21 novembre 2008, Metz, France.

[CHE 08c] Chevallereau, B., Bernard, A., Mévellec, P., 2008, *Contribution des nouvelles approches de modélisation à la durabilité des applications*, 3ème Forum Académique de l'AFIS, 2-4 décembre 2008, Nîmes, France, Poster.

[CHE 09c] Chevallereau, B., Bernard, A., Mévellec, P., 2009, *Améliorer les performances de l'industrie logicielle par une meilleure compréhension des besoins*, 3ème journées nationales du GDR MACS, 17-18 mars 2009, Angers, France.

[CHE 09d] Chevallereau, B., Bernard, A., Mévellec, P., 2009, *Ingénierie dirigée par les modèles pendant la phase de spécification du besoin*, 5ème journées sur l'Ingénierie Dirigée par les Modèles, 25-26 mars 2009, Nancy, France.

[DAA 09] Daaboul, J., Xu, Y., Vergara, V., Le Duigou, J., Chevallereau, B., Rauffet, P., Laroche, F., Da Cunha, C., Bernard, A., 2009, *Amélioration de la performance industrielle par l'ingénierie numérique*, 11ème colloque national AIP Primeca, 22-24 avril 2009, La Plagne, France.

[CHE 09e] Chevallereau, B., 2009, *Comment concevoir des télé procédures par des experts fonctionnels?*, 12ème journées STP du GDR MACS, 28-29 octobre 2009, Annecy, France.

[CHE 10a] Chevallereau, B., Bernard, A., Mévellec, P., 2010, *Contribution des nouvelles approches de modélisation à la durabilité des applications*, Séminaire de l'école doctorale SPIGA, 28 mai 2010, Le Mans, France.

[CHE 10b] Chevallereau, B., Bernard, A., 2010, *Concevoir un système d'information hospitalier évolutif, ouvert et adapté grâce à processus de conception orienté modèle*, Conférence Gestion et Ingénierie des SystèmEs Hospitaliers, 2-4 septembre 2010, Clermont-Ferrand, France.

B. Bibliographie externe

[AGU 08] Aguirre-Urreta, M.I., Marakas, G.M., 2008, *Comparing conceptual modeling techniques: a critical review of the EER vs. OO empirical litterature*, SIGMIS Database, vol. 39, pp. 9-32.

[AND 98] Andro, T., Chauvet, J-M., 1998, *Objets métier*, Édition Eyrolles, Paris.

[ARB 10] Arborio, A.-M., Fournier, P., 2010, *L'observation directe: L'enquête et ses méthode*, 3ème édition, Édition Armand Colin.

[ASU 86] Aho, A.V., Sethi, R., Ullman, J.D., 1986, *Compilers : Principles, Techniques, and Tools*, Addison-Wesley Longman Publishing Co., Boston.

[ATL 05] INRIA ATLAS, 2005, *KM3 : Kernel MetaMetaModel*. Rapport technique, http://www.eclipse.org/gmt/am3/km3/doc/KernelMetaMetaModel%5Bv00.06%5D.pdf, dernier accès : Juillet 2010.

[BAC 02] Bacha, R., 2002, *De la gestion des données techniques pour l'ingénierie de production : Référentiel du domaine et cadre méthodologique pour l'ingénierie des systèmes d'information techniques en entreprise*, Thèse de doctorat, École Centrale Paris.

[BAS 91] Bastien, C., 1991, *Validation de critères ergonomiques pour l'évaluation d'interfaces utilisateurs*, Rapports de recherche, Éditions INRIA.

[BEC 04] Beck, K., Andres, C., 2004, *Extreme Programming Explained: Embrace Change*, 2nde édition, Addison-Wesley Professional.

[BEZ 03] Bézivin, J., 2003, *La transformation de modèles*, École d'Été d'Informatique CEA/EDF/INRIA, 16-27 juin 2003, Saint-Lambert-des-Bois, France.

[BEZ 04] Bézivin, J., 2004, *Sur les principes de base de l'ingénierie des modèles*, L'Objet, vol. 10, pp. 145-157.

[BLA 05] Blanc, X., 2005*, MDA en action - Ingénierie logicielle guidée par les modèles*, 1ère édition, Éditions Eyrolles, Paris.

[BLA 10] Blanchet, A., Gotman, A., 2010, *L'entretien: L'enquête et ses méthodes*, 2ème édition, Édition Armand Colin.

[BOE 81] Boehm, B., *Software Engineering Economics*, Édition Prentice-Hall, New-Jersey.

[BOE 84] Boehm, B., Gray, T.E., Seewaldt, T., *Prototyping vs. specifying : a multi-project experiment*, 7th International Conference on Software Engineering (ICSE), 26-29 mars 1984, Orlando, Etats-Unis.

[BRAN 03] Brangier, E., Barcenilla, J., 2003, *Concevoir un produit facile à utiliser*, Éditions d'organisation, Paris.

[BRAN 04] Brangier, E., Vallery, G., 2004, *Aspects psychologiques et organisationnels du développement des nouvelles technologies de la communication et de l'information, Les dimensions humaines du travail : théories et pratiques de la psychologie du travail et des organisations*, Presses Universitaires de Nancy, pp. 213-250.

[BRE 91] Brenier, H., 2001, *Les spécifications fonctionnelles : automatismes industriels et temps réel*, Éditions Dunod, Paris.

[BUD 03] Budinsky, F., Steinberg, D., Ellersick, R., 2003, *Eclipse Modeling Framework : A Developer's Guide*, Addison-Wesley Professional, Boston.

[CAL 90] Calvez, J.P., 1990, *Spécification et conception des systèmes : une méthodologie*, Éditions Masson, Paris.

[CAS 01] Castro, J., Kolp, M., Mylopoulos, J., 2001, *A requirements driven-development methodology*, 13th International Conference on Advanced Information Systems Engineering (CAiSE), 4-8 juin 2001, Interlaken, Suisse.

[CAS 02] Castro, J., Kolp, M, Mylopoulos, J., 2002, *Towards requirements-driven information systems engineering : the Tropos project*, Information Systems, vol. 27, pp. 365-389.

[CHE 76] Chen, P., 1976, *The Entity-Relationship Model – Toward a unified view of data*, ACM Transactions on Database Systems, vol. 1, pp. 9-36.

[CHU 00] Chung, L., Nixon, B., Yu, E., Mylopoulos, J., 2000, *Non-Functional Requirements in Software Engineering*, The Kluwer International Series in Software Engineering, vol. 5.

[CLA 04] Clark, T., Evans, A., Sammut, P., Willans, J., 2004, *Applied Metamodelling: A Foundation for Language Driven Development*, version 0.1, http://www.uio.no/studier/emner/matnat/ifi/INF5120/v06/undervisningsmateriale/AppliedMetamodelling.pdf, dernier accès : Juillet 2010.

[COM 08] Combemale, B., 2008, *Approche de métamodélisation pour la simulation et la vérification de modèle*, Thèse de doctorat, Institut National Polytechnique de Toulouse.

[COO 01] Cooke, D., Gelman, L., Peterson, W., 2001, *ERP Trends*, The Conference Board.

[DAM 06] Damas, C., Lambeau, B., Lamsweerde, A.V., 2006, *Scenarios, goals, and state machines: a win-win partnership for model synthesis*, 14th ACM SIGSOFT international symposium on Foundations of Software Engineering (FSE), 5-11 novembre 2006, Portland, Etats-Unis.

[DAR 93] Dardenne, A., Lamsweerde, A. V., Fickas, S., 1993, *Goal-Directed Requirements Acquisition*, Science of Computer Programming, vol. 20, pp. 3-50.

[DEU 74] Deutcher, I., 1974, *What we say/What we do*, The sociological Quaterly, vol. 5, pp. 457-461.

[EHR 04] Ehrig, K., Ermel, C., Hänsgen, S., Taentzer, G., 2004, *Towards Graph Transformation based Generation of Visual Editors using Eclipse*, Workshop Visual Languages and Formal Methods (VLFM), 30 septembre 2004, Rome, Italie.

[EUR 96] European Software Institute, 1996, *Report USV EUR 2.1 - ESPITI Project*, European User Survey Analysis.

[FAR 06] Farail, P., Gaufillet, P., Canals, A., Camus, C.L., Sciamma, D., Michel, P., Crégut, X., Pantel, M., 2006, *The TOPCASED project : a Toolkit in OPen source for Critical Aeronautic SystEms Design*, 3rd European Congress Embedded Real Time Software (ERTS), 25-27 janvier 2006, Toulouse, France.

[GAB 98] Gabay, J., 1998, *Merise vers OMT et UML: un guide complet avec études de cas*, 3ème édition, InterEditions, Paris.

[GAB 04] Gabay, J., 2004, *Merise et UML : Pour la modélisation des systèmes d'information*, 5ème édition, Édition Dunod/01 Informatique, Paris.

[GAM 95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995, *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Professional Computing Series, Boston.

[GAU 89] Gause, D., Weinberg, G., 1989, *Exploring Requirements : Quality Before Design*, Dorset House Publishing Company.

[GER 02] Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A., 2002, *Transformation : The Missing Link of MDA*, 1st International Conference on Graph Transformation (ICGT), 7-12 octobre 2002, Barcelone, Espagne.

[GIU 02] Giunchiglia, F., Mylopoulos, J., Perini, A., 2002, *The tropos software development methodology: processes, models and diagrams*, 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 15-19 juillet 2002, Bologne, Italie.

[GLA 06] Glass, R., 2006, *The Standish report: does it really describe a software crisis?*, Communications of the ACM, vol. 49, pp. 15-16.

[GRA 07] Grari, M., 2007, *Principes et états de l'art de l'approche MDA et applications pour des plates-formes PHP orienté 3-tiers*, rapport de mémoire, Université Mohammed Premier.

[HOC 01] Hochheiser, H., Schneiderman, B., 2001, *Universal usability statements: Marking the trail for all users*, ACM interactions journal, vol. 8, pp. 16-18.

[ISO 98] International Organization for Standardization (ISO), 1998, *ISO 9241-11:1998 : Exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation (TEV) -- Partie 11: Lignes directrices relatives à l'utilisabilité*.

[ISO 99] International Organization for Standardization (ISO), 1999, *ISO 13407:1999 : Processus de conception centrée sur l'opérateur humain pour les systèmes interactifs*.

[ISO 10] International Organization for Standardization (ISO), 2010, *ISO 9241-210:2010 : Ergonomie de l'interaction homme-système -- Partie 210: Conception centrée sur l'opérateur humain pour les systèmes interactifs*.

[JAC 95] Jackson, M., 1995, *Software Requirements and Specifications – A Lexicon of Practice, Principles and Pejudices*, Addison-Wesley, Boston.

[JAU 94] Jaulent, P., *Génie logiciel : les méthodes : SADT, SA, E-A, SA-RT, SYS_P_O, OOD, HOOD...*, 3ème édition, Édition Armand Colin, Paris.

[JOH 95] Johnson, J., 1995, *Chaos: The Dollar Drain of IT Project Failures*, Application Development Trends, pp. 41-47.

[JOU 05] Jouault, F., Kurtev, I., 2005, *Transforming Models With ATL*, Satellite Events at the MoDELS Conference, 2-7 octobre 2005, Montego Bay, Jamaïque.

[JOU 06a] Jouault, F., Bézivin, J., 2006, *KM3 : a DSL for Metamodel Specification*, 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS), 14-16 juin 2006, Bologne, Italie.

[JOU 06b] Jouault, F., Bézivin, J., Kurtev, I., 2006, *TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering*, 5th International Conference on Generative Programming and Component Engineering (GPCE), 22-26 octobre 2006, Portland, USA.

[KLE 03] Kleppe, A., Warmer, J., Bast, W., 2003, *MDA Explained : The Model Driven Architecture – Practice and Promise*, Addison-Wesley Professional, Boston.

[KOL93] Kolski, C., 1993, *Ingénierie des interfaces homme-machine, conception et évaluation*, Édition Hermès, Paris,

[KOT 98] Kotonya, G., Sommerville, I., 1998, *Requirements engineering processes and techniques*, John Wiley & sons.

[KRO 05] Krob, D., 2005, *La mise en œuvre des systèmes d'information: quelques éléments d'analyse systémique*, Journée de la Chaire « Ingénierie des Systèmes Complexes » de l'École Polytechnique, 31 mars 2005, Palaiseau, France.

[LAM 01] Lamsweerde, A.V., 2001, *Goal-oriented requirements engineering : a guided tour*, 5th International Symposium on Requirements Engineering (RE), 27-31 août 2001, Toronto, Canada.

[LAM 03] Lamsweerde, A.V., Letier, E., 2003, *From object orientation to goal orientation : a paradigm shift for requirements engineering*, Radical Innovations of Software & System Engineering (LNCS).

[LED 01] Ledeczi, A., Marioti, A., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., Volgyesi, P., *The Generic Modeling Environment*, IEEE International Workshop on Intelligent Signal Processing (WISP), 24-25 mai 2001, Budapest, Hongrie.

[LEU 02] Leung, H., Fan, Z., 2002, *Software Cost Estimation, Dans : Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Company, River Edge.

[MAR 91] Martin, J., 1991, *Rapid Application Development*, MacMillan, New York.

[MID 03] The Middleware Company, 2003, *Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach – Productivity Analysis*, http://www.omg.org/mda/mda_files/MDA_Comparison-TMC_final.pdf, Dernier accès : Juillet 2010.

[MIL 01] Miller, J., Mukerji, J., 2001, *MDA Guide Version 1.0*, http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf, Dernier accès : Juillet 2010.

[MIL 03] Miller, J., Mukerji, J., 2003, *MDA Guide Version 1.0.1*, http://www.omg.org/cgi-bin/doc?omg/03-06-01, Dernier accès : Juillet 2010.

[MIL 06] Mille, A., Prié, Y., 2006, *Une théorie de la trace informatique pour faciliter l'adaptation dans la confrontation logique d'utilisation/logique de conception*, 13èmes Rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels, 22-27 janvier 2006, Megève, France.

[MIR 06] Miralles, A., 2007, *Ingénierie des modèles pour les applications environnementales*, Thèse de doctorat, Université de Montpellier II.

[MIS 05] Misra, S., Kumar, V., Kumar, U., 2005, *Goal-oriented or scenario-based requirements engineering technique – what should a practitioner select ?*, 18th Annual Canadian Conference on Electrical and Computer Engineering (CCECE), 1-4 mai 2001, Saskatchewan, Canada.

[MOO 04] Moore, W., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P., 2004, *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*, 1ère édition, IBM.

[MOR 06] Morley, C., 2006, *Management d'un projet système d'information : Principes, techniques, mise en œuvre et outils*, 5ème édition, Édition Dunod, Paris.

[MOU 07] Mouloudi, A., 2007, *Intégration des besoins des utilisateurs pour la conception de système d'information voyageur*, Thèse de doctorat, Université de Technologie de Compiègne.

[MUL 98] Muller, M.J., Matheson, L., Page, C., Gallup, R., 1998, *Methods & tools: participatory heuristic évaluation*, Journal of interactions, vol. 5, issue 5, pp. 13-18.

[MUL 05a] Muller, P.-A., Fleurey, F., Jézéquel, J.-M., 2005, *Weaving Executability into Object-Oriented Meta-Languages*, ACM/IEEE 8th International Conference On Model Driven Engineering Languages And Systems (MoDELS), 2-7 octobre 2005, Montego Bay, Jamaïque.

[MUL 05b] Muller, P.-A., Fondement, F., Fleurey, F., Hassenforder, M., Schekenburger, M., Gérard, S., Jézéquel, J.-M., 2005, *Model-driven analysis and synthesis of textual concrete syntax*, Journal of Software and Systems Modeling (SoSyM), vol. 7, pp. 423-442.

[MYL 92] Mylopoulos, J., Chung, L., Nixon, B., 1992, *Representing and Using Non-Functional Requirements : A Process-Oriented Approach*, IEEE Transactions on Software Engineering, vol. 18, pp. 483-497.

[NIE 90] Nielsen, J., Molich, R., 1990, *Heuristic evaluation of user interfaces*, SIGCHI Conference on Human Factors in Computing Systems (CHI), 1-5 avril 1990, Seattle, Etats-Unis.

[NIE 92] Nielsen, J., 1992, *The Usability Engineering Life Cycle*, IEEE Computer, vol. 25, pp. 12-22.

[NIE 93] Nielsen, J., 1993, *Usability Engineering*, Éditions Academic Press.

[NOR 99] Norman, D., 1999, *Affordance, conventions, and design*, ACM interactions journal, vol. 6, pp. 38-43.

[OMG 02] OMG, 2002, *MOF 2.0 Query/Views/Transformations*, http://www.omg.org/spec/QVT/1.0/PDF/, Dernier accès : Juillet 2010.

[OMG 06] OMG, 2006, *Meta Object Facility (MOF) 2.0 Core Specification*, http://www.omg.org/spec/MOF/2.0/PDF/, Dernier accès : Juillet 2010.

[OMG 07] OMG, 2007, *Unified Modeling Language (UML) 2.1.2 Superstructure*, http://www.omg.org/cgi-bin/doc?formal/09-02-03.pdf, Dernier accès : Juillet 2010.

[PAT 08a] Patig, S., 2008, *Preparing Meta-Analysis of Metamodel Understandab*ility, 11th International Conference on Model Driven Engineering Languages and Systems (MODELS), 28 septembre – 3 octobre 2008, Toulouse, France.

[PAT 08b] Patig, S., 2008, *A Practical Guide to Testing the Understandability of Notations*, 5th AsiaPacific Conference on Conceptual Modelling (APCCM), 22-25 janvier 2008, Wollongong, Australie.

[POT 99] Potts, C., 1999, *ScenIC : A strategy for inquiry-driven requirements determination*, 4th International Symposium on Requirements Engineering (RE), 7-11 juin 1999, Limerick, Irlande.

[RIE 00] Riecken, D., 2000, *Introduction: personalized views of personalization*, Communications of the ACM, vol. 43, pp. 26-28.

[ROB 02] Robbins-Gioia LLC, 2002, *ERP Survey Results Point to Need For Higher Implementation Success*.

[ROS 02] Rosson, M.B., Carroll, J.M., 2002, *Usability engineering : Scenario-based development of human-computer interaction*, Édition Morgan Kaufmann, San Francisco.

[ROL 07] Rolland, C., 2007, *Capturing System Intentionality with Maps*, Conceptual Modelling in Information Systems Engineering, pp. 141-158.

[SCH 00] Schneiderman, B., 2000, *Universal usability*, Communications of the ACM, vol. 43, pp. 84-91.

[SIN 08] Singly F.D., 2005, *L'enquête et ses méthodes : Le questionnaire*, 2ème édition, Éditeur Armand Colin.

[SOM 97] Sommerville, I., Sawyer, P., 1997, *Requirements engineering : A good practice*, John Wiley & sons.

[STA 94] Standish Group, 1994, *The Chaos Report*.

[STA 97] Stapleton, J., 1997, *DSDM, Dynamic System Development Method*, Addison-Wesley, Harlow.

[SUT 98] Sutcliffe, A., Ryan, M., 1998, *Experience with SCRAM : a SCenario Requirements Analysis Method*, 3th International Conference on Requirements Engineering (ICRE), 6-10 avril 1998, Colorado Springs, Etats-Unis.

[SUT 03] Sutcliffe, A., 2003, *Scenario-Based Requirements Engineering*, 11th International Requirements Engineering Conference (RE), 8-12 septembre 2003, Monterey, Etats-Unis.

[TOP 02] Topi, H, Ramesh, V., 2002, *Human Factors Research on Data Modeling: A Review of Prior Research, An Extended Framework and Future Research Directions*, Journal of Database Management, vol. 13, pp. 3-19.

[VAU 03] Vaudrin, F., 2003, *Conception d'un système d'information grand public sur les conditions routières*, Mémoire de maîtrise, Université du Québec à Chicoutimi.

[YU 97] Yu, E., 1997, *Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering*, 3rd International Symposium on Requirements Engineering (RE), 6-8 janvier 1997, Annapolis, États-Unis.

Annexes

Annexe A Modèles définis dans le tutoriel KAOS pour un système d'élévateur.

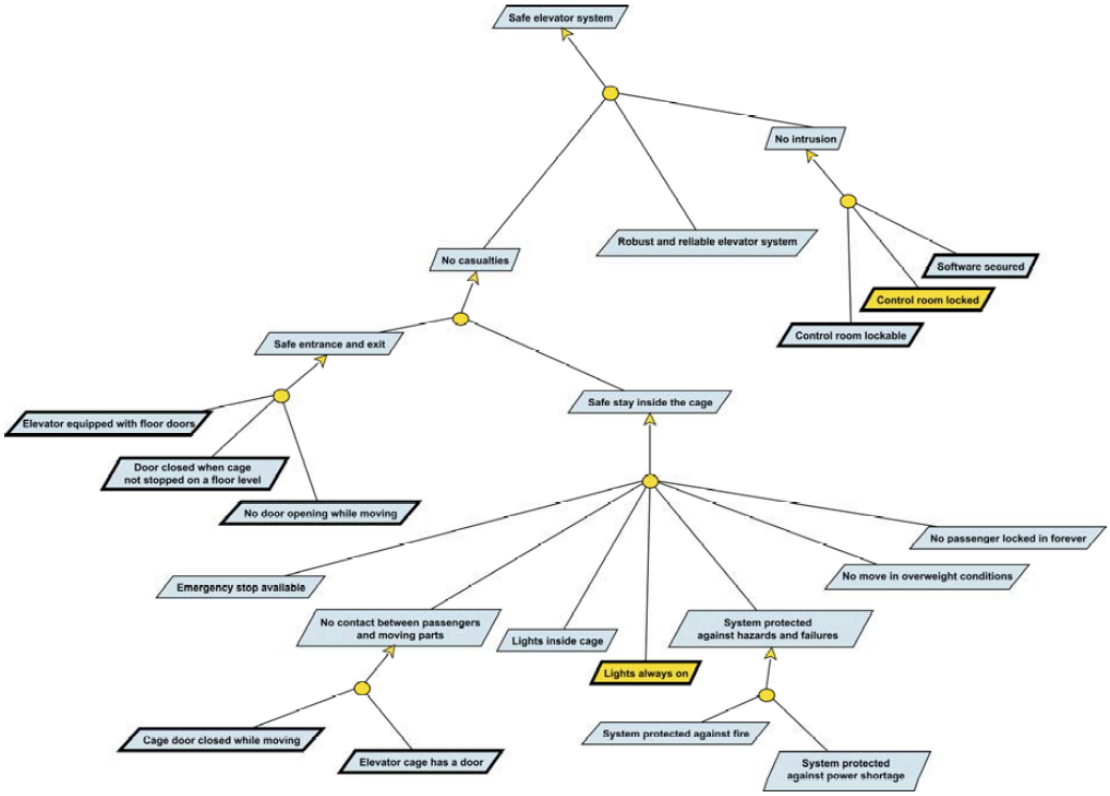


Figure A-1 : Exemple de modèle des buts KAOS

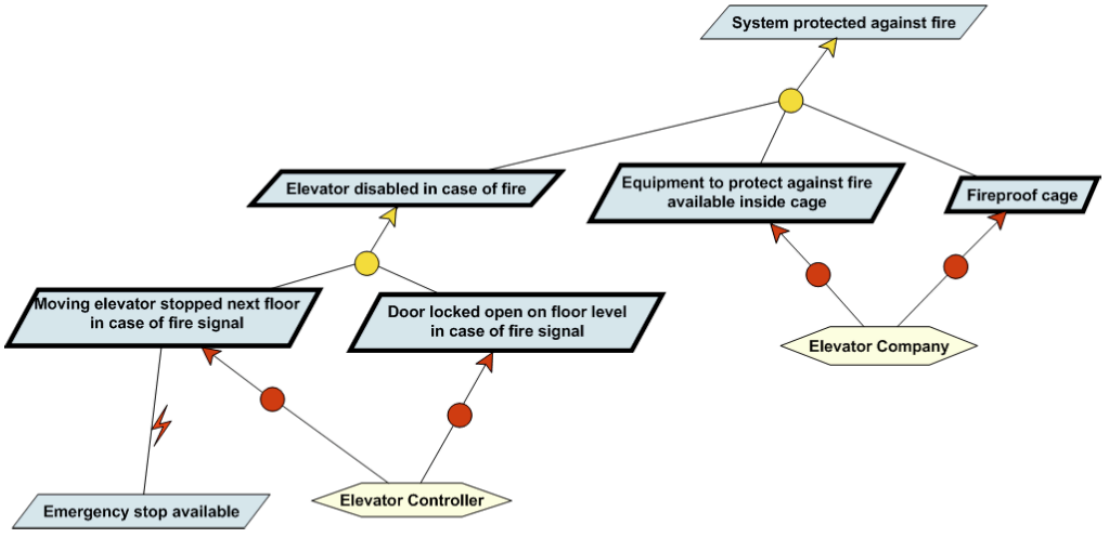


Figure A-2 : Exemple de modèle des responsabilités KAOS

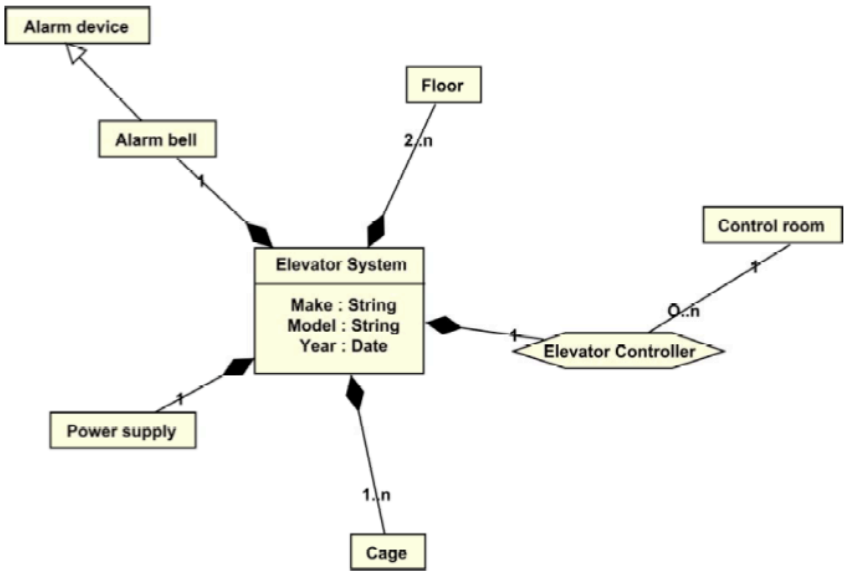


Figure A-3 : Exemple de modèle objet KAOS

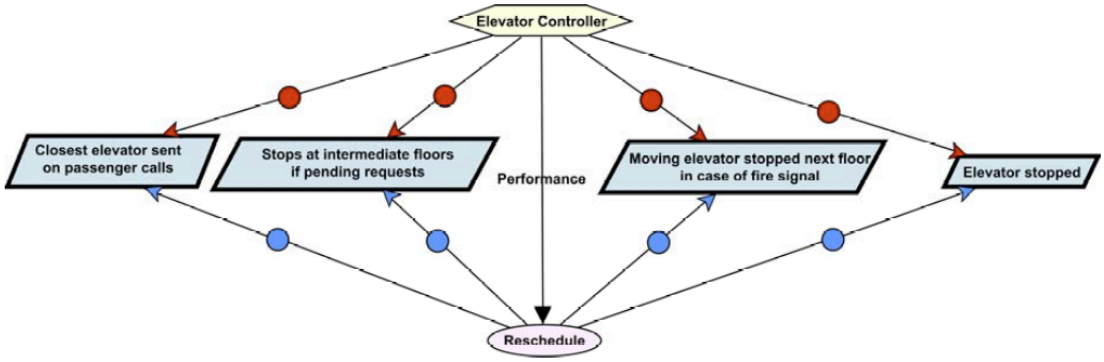


Figure A-4 : Exemple de modèles d'opération KAOS

Annexe B Exemples de modèles utilisant le langage i*

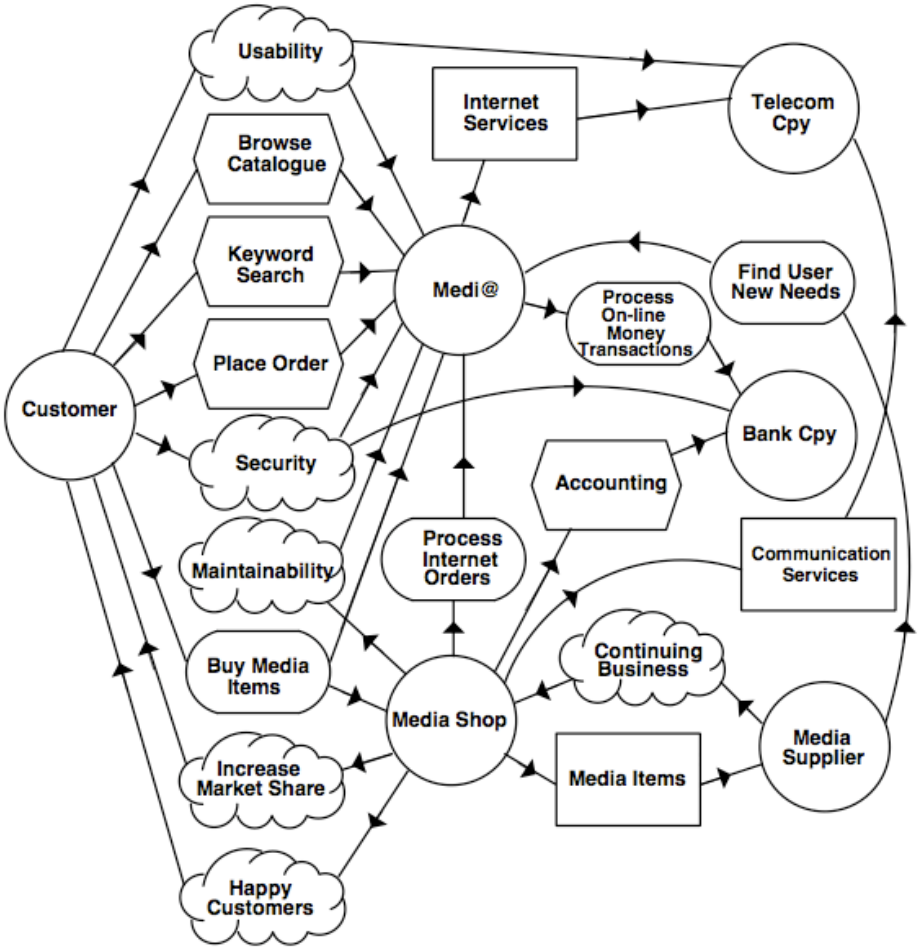


Figure B-5 : Exemple de modèle SD proposé par la méthodologie i* [CAS 01]

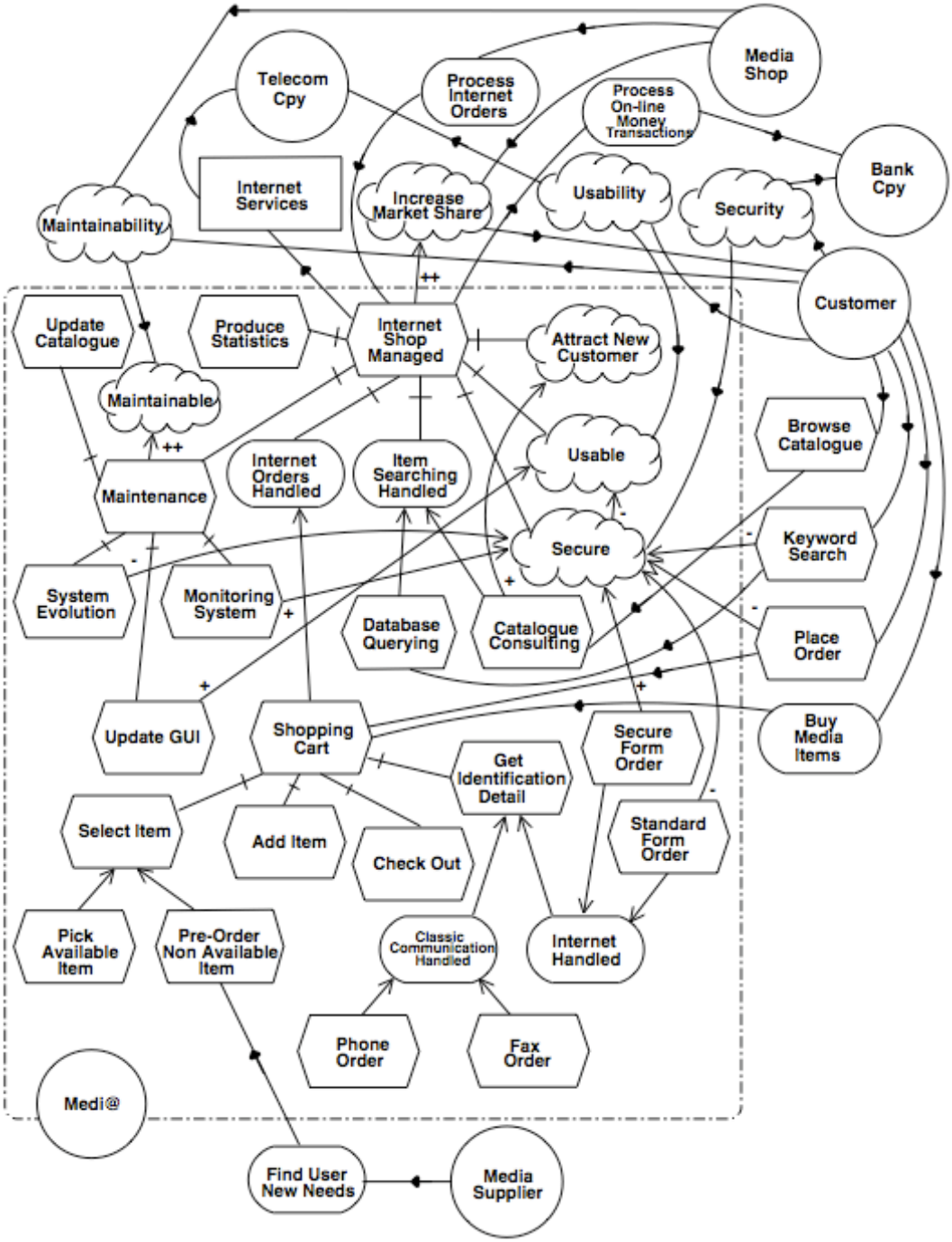
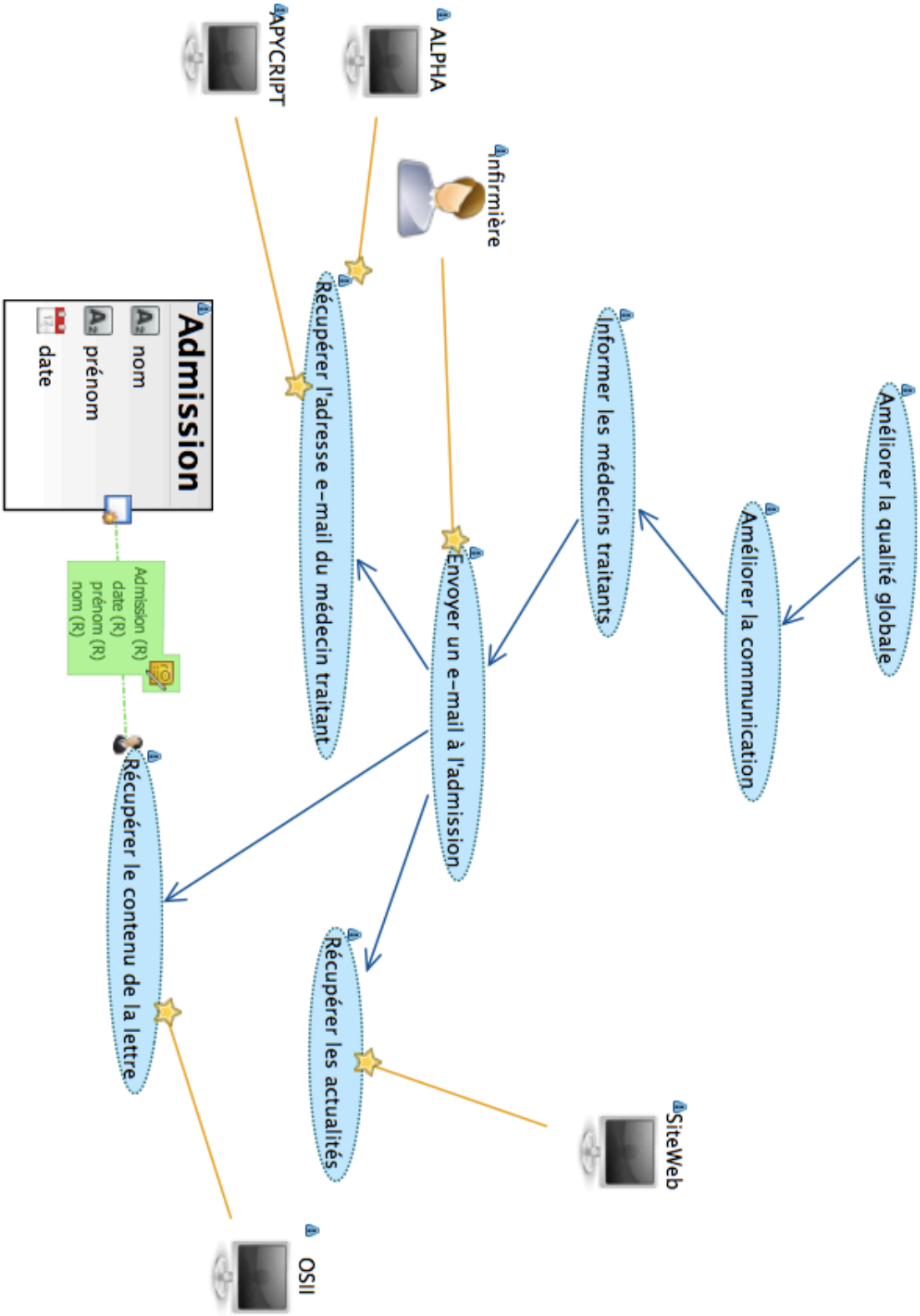
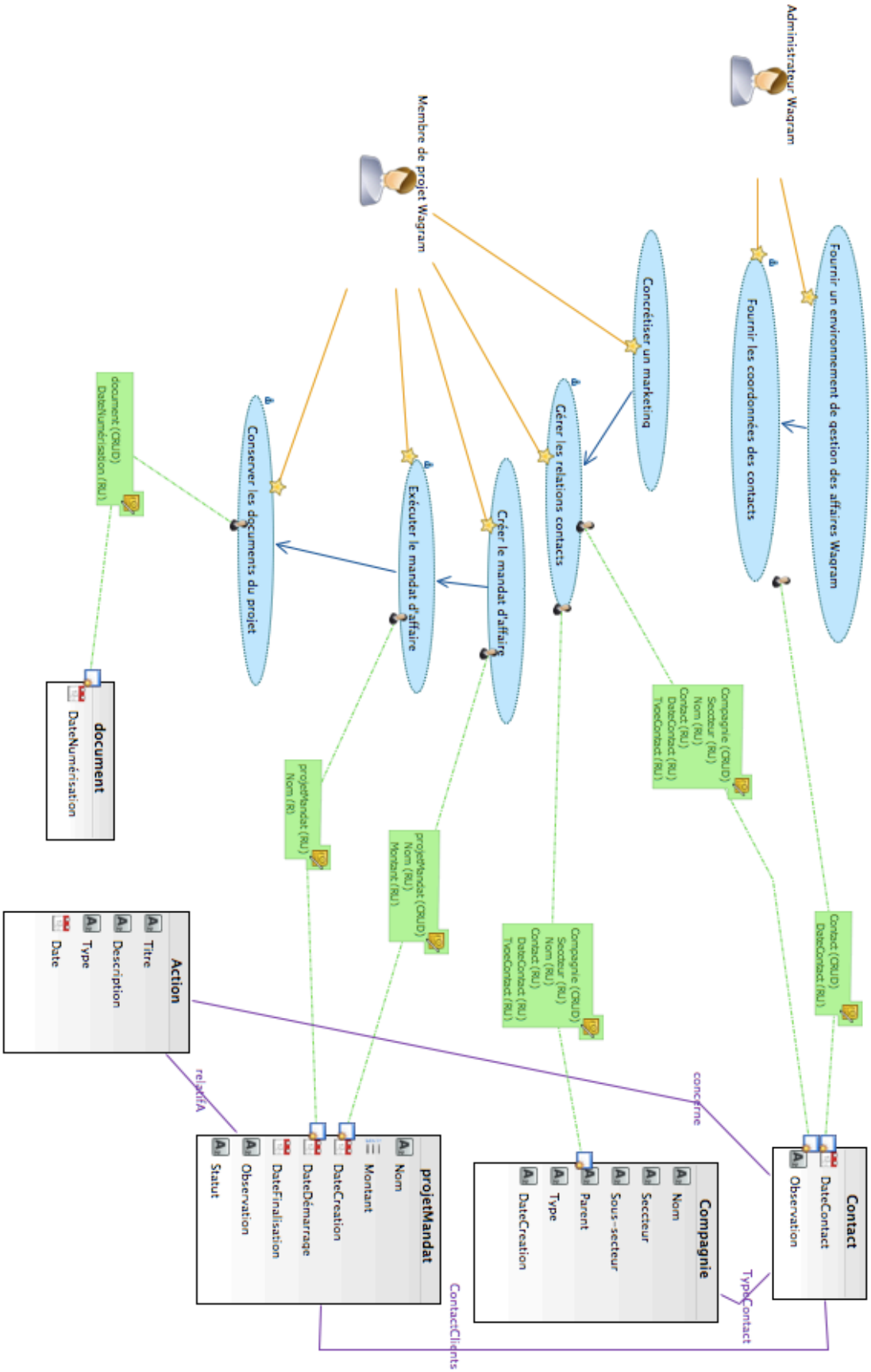


Figure B-6 : Exemple de modèle SR proposé par la méthodologie i* [CAS 01]

Annexe C Modèle Saint Augustin



Annexe D Modèle Wagram



Annexe E Transformation de modèles

Listing 6 : Transformation ATL Req2Diagnostic

```
module RWM2Problem; -- Module Template
create OUT : Diagnostic from IN : RWM;

helper context RWM!Goal def : getAllSubGoals() : Sequence(RWM!Goal) =
	Sequence{self}->union(self.subgoals->collect(goal |
goal.getAllSubGoals())->flatten());

-------------------------------------------------
-- BasicElement
-------------------------------------------------

helper context RWM!BasicElement def : checkElement() :
Sequence(Diagnostic!Problem) =
	Sequence{self.checkElementName(), self.checkElementDocumentation()};

helper context RWM!BasicElement def : checkElementName() :
Diagnostic!Problem =
	if (self.name = '' or self.name = OclUndefined) then
	   thisModule.createProblem(#error,
self.oclType().toString(),self.name,'Name empty.')
	else
		OclUndefined
	endif;

helper context RWM!BasicElement def : checkElementDocumentation() :
Diagnostic!Problem =
	if (self.documentation = '' or self.documentation = OclUndefined)
then
	   thisModule.createProblem(#warning,
self.oclType().toString(),self.name,'Documentation empty.')
	else
		OclUndefined
	endif;

-------------------------------------------------
-- Agent
-------------------------------------------------

helper context RWM!Agent def : checkAgent() : Sequence(Diagnostic!Problem)
=
	Sequence{self.checkAgentWithoutResponsibilty(),self.checkAgentWithSam
eResponsibilty(),self.checkAgentWithRedundantResponsibilty()};

helper context RWM!Agent def : checkAgentWithoutResponsibilty() :
Diagnostic!Problem =
	if (self.isResponsible->size() = 0) then
	   thisModule.createProblem(#critical,
self.oclType().toString(),self.name,'Agent without responsibility.')
	else
```

```
            OclUndefined
      endif;

helper context RWM!Agent def : checkAgentWithSameResponsibilty() :
Diagnostic!Problem =
      if (RWM!Agent->allInstances()->excluding(self)->select(a |
a.isResponsible->includesAll(self.isResponsible) and self.isResponsible-
>includesAll(a.isResponsible))->size() > 0) then
         thisModule.createProblem(#warning,
self.oclType().toString(),self.name,RWM!Agent->allInstances()->select(a |
a.isResponsible->includesAll(self.isResponsible) and self.isResponsible-
>includesAll(a.isResponsible)).toString()+' have same responsibility.')
      else
            OclUndefined
      endif;

helper context RWM!Agent def : checkAgentWithRedundantResponsibilty() :
Diagnostic!Problem =
    self.isResponsible->iterate(goal; problem : Diagnostic!Problem =
OclUndefined |
          if (self.isResponsible->collect(goal | goal.getAllSubGoals()-
>excluding(goal))->flatten()->asSet()->includes(goal)) then
                thisModule.createProblem(#critic,
self.oclType().toString(),goal.name,'The responsibility on this goal is
useless for the user '+self.name+'.')
          else
              OclUndefined
          endif
     );

--------------------------------------------------
-- Goal
--------------------------------------------------

helper context RWM!Goal def : checkGoal() : Sequence(Diagnostic!Problem) =
      Sequence{self.checkGoalPrivilege()};

helper context RWM!Goal def : checkGoalPrivilege() : Diagnostic!Problem =
      if (self.getAllSubGoals()->collect(g | g.privilegeGroup)->flatten()-
>excluding(OclUndefined)->collect(g | g.privileges)->size() > 0) then
         thisModule.createProblem(#critic,
self.oclType().toString(),self.name,'No privilege.')
      else
            OclUndefined
      endif;

helper def : checkMainGoal() : Diagnostic!Problem =
      if (RWM!Goal->allInstances()->select(goal | RWM!Goal->allInstances()-
>select(g | g.subgoals->includes(goal))->size() = 0)->size() > 1) then
         thisModule.createProblem(#error, 'RWM!Goal','','More than one
main goal : '+RWM!Goal->allInstances()->select(goal | RWM!Goal-
>allInstances()->select(g | g.subgoals->includes(goal))->size() =
0).toString()+'.')
      else
            OclUndefined
      endif;
```

```
-----------------------------------------------
-- Entity
-----------------------------------------------

helper context RWM!Entity def : checkEntity() :
Sequence(Diagnostic!Problem) =
	Sequence{self.checkEntityProperties(), self.checkEntityPrivilege(),
self.checkEntityPrivilege_2(), self.checkEntityPrivilege_3()};

helper context RWM!Entity def : checkEntityProperties() :
Diagnostic!Problem =
	if (self.attributes->size() = 0 and RWM!RelationShip-
>allInstances()->select(r | r.source = self or r.target = self)->size() =
0) then
		thisModule.createProblem(#critic,
self.oclType().toString(),self.name,'No attribute and relationship.')
	else
		OclUndefined
	endif;

helper context RWM!Entity def : checkEntityPrivilege() :
Diagnostic!Problem =
	if (RWM!Privilege->allInstances()->select(p | p.element = self)-
>size() = 0) then
		thisModule.createProblem(#critic,
self.oclType().toString(),self.name,'No privilege on this element.')
	else
		OclUndefined
	endif;

helper context RWM!Entity def : checkEntityPrivilege_2() :
Diagnostic!Problem =
	if (RWM!Privilege->allInstances()->select(p | p.element = self and
p.category = #update)->size() > 1 and
		RWM!Privilege->allInstances()->select(p | p.element = self
and p.category = #read)->size() = 0) then
		thisModule.createProblem(#critic,
self.oclType().toString(),self.name,'Update available on this element but
not read.')
	else
		OclUndefined
	endif;

helper context RWM!Entity def : checkEntityPrivilege_3() :
Diagnostic!Problem =
	if (RWM!Privilege->allInstances()->select(p | p.element = self and
p.category = #delete)->size() > 1 and
		RWM!Privilege->allInstances()->select(p | p.element = self
and p.category = #read)->size() = 0) then
		thisModule.createProblem(#critic,
self.oclType().toString(),self.name,'Delete available on this element but
not read.')
	else
		OclUndefined
	endif;

-----------------------------------------------
```

```
-- Attribute
------------------------------------------------

helper context RWM!Attribute def : checkAttribute() :
Sequence(Diagnostic!Problem) =
	Sequence{self.checkAttributePrivilege(),
self.checkAttributePrivilege_2()};

helper context RWM!Attribute def : checkAttributePrivilege() :
Diagnostic!Problem =
	if (RWM!Privilege->allInstances()->select(p | p.element = self)-
>size() = 0) then
		thisModule.createProblem(#critic,
self.oclType().toString(),self.name,'No privilege on this element.')
	else
		OclUndefined
	endif;

helper context RWM!Attribute def : checkAttributePrivilege_2() :
Diagnostic!Problem =
	if (RWM!Privilege->allInstances()->select(p | p.element = self and
p.category = #update)->size() > 1 and
		RWM!Privilege->allInstances()->select(p | p.element = self
and p.category = #read)->size() = 0) then
		thisModule.createProblem(#critic,
self.oclType().toString(),self.name,'Update available on this element but
not read.')
	else
		OclUndefined
	endif;

------------------------------------------------
-- RelationShip
------------------------------------------------

helper context RWM!RelationShip def : checkRelationShip() :
Sequence(Diagnostic!Problem) =
	Sequence{self.checkRelationShipPrivilege(),
self.checkRelationShipPrivilege_2(),self.checkRelationShipPrivilege_3()};

helper context RWM!RelationShip def : checkRelationShipPrivilege() :
Diagnostic!Problem =
	if (RWM!Privilege->allInstances()->select(p | p.element = self)-
>size() = 0) then
		thisModule.createProblem(#critic,
self.oclType().toString(),self.name,'No privilege on this element.')
	else
		OclUndefined
	endif;

helper context RWM!RelationShip def : checkRelationShipPrivilege_2() :
Diagnostic!Problem =
	if (RWM!Privilege->allInstances()->select(p | p.element = self and
p.category = #update)->size() > 1 and
		RWM!Privilege->allInstances()->select(p | p.element = self
and p.category = #read)->size() = 0) then
		thisModule.createProblem(#critic,
```

```
self.oclType().toString(),self.name,'Update available on this element but
not read.')
	else
		OclUndefined
	endif;

helper context RWM!RelationShip def : checkRelationShipPrivilege_3() :
Diagnostic!Problem =
	if (self.target = OclUndefined or self.source = OclUndefined) then
	   thisModule.createProblem(#error,
self.oclType().toString(),self.name,'No valid.')
	else
		OclUndefined
	endif;

-----------------------------------------------

rule Diagnostic {
	from s : RWM!RequirementsDefinition
	to t : Diagnostic!Diagnostic (
		)
    do {
		--BasicElement
	for (e in RWM!BasicElement->allInstances()) {
		e.checkElement();
	}
		--Agent
	for (e in RWM!Agent->allInstances()) {
		e.checkAgent();
	}
		--Goal
		thisModule.checkMainGoal();
	for (e in RWM!Goal->allInstances()) {
		e.checkGoal();
	}
	--Entity
	for (e in RWM!Entity->allInstances()) {
		e.checkEntity();
	}
	--Attribute
	for (e in RWM!Attribute->allInstances()) {
		e.checkAttribute();
	}
	--RelationShip
	for (e in RWM!RelationShip->allInstances()) {
		e.checkRelationShip();
	}
	t.problem <- Diagnostic!Problem->allInstances();
    }
}

rule createProblem(_severity : Diagnostic!Severity, _elementType : String,
_elementName : String, _description : String) {
	to t : Diagnostic!Problem (
		severity <- _severity,
		elementType <- _elementType,
		elementName <- _elementName,
```

```
            description <- _description
        )
        do {
            t;
        }
}
```

Listing 7 : Transformation ATL Req2GoalList

```
-- @atlcompiler atl2006
module MindMap; -- Module Template
create OUT : MindMap from IN : RWM;

helper context String def : normalize() : String =
	self.regexReplaceAll('<([a-zA-Z0-
9]*)>(.*)</\\1>','$2').regexReplaceAll('<([a-zA-Z0-9]*)>','')
		.regexReplaceAll('&lt;([a-zA-Z0-
9]*)>(.*)&lt;/\\1>','$2').regexReplaceAll('&lt;([a-zA-Z0-9]*)>','')

	.regexReplaceAll('\\\\(.)','$1').regexReplaceAll('&','&amp;').regexRep
laceAll('"','&quot;');

rule definition {
  from s: RWM!RequirementsDefinition
  to t : MindMap!"Map" (
	node <- t_node
	),
	t_node : MindMap!Node (
		text <- 'Liste des objectifs',
		sub <- RWM!Goal->allInstances()->reject(goal |
goal.refImmediateComposite().oclIsTypeOf(RWM!Goal))
	)
}

rule goal {
	from s : RWM!Goal
	to t : MindMap!Node (
		text <- s.name,
		sub <- s.subgoals
	)
}

rule goalWithDocumentation extends goal {
	from s : RWM!Goal (s.documentation.normalize().size() > 0)
	to t : MindMap!Node (
		sub <- Sequence{description}->union(s.subgoals)
	),
	description : MindMap!Node (
		text <- s.documentation.normalize(),
		presentation <- font_italic
	),
	font_italic : MindMap!NodePresentation (
		bold <- false,
		italic <- true,
		size <- 10,
		name <- 'SansSerif'
	)
}
```

Listing 8 : Transformation ATL Req2GoalListByAgent

```
-- @atlcompiler atl2006
module MindMap; -- Module Template
create OUT : MindMap from IN : RWM;

helper context String def : normalize() : String =
	self.regexReplaceAll('<([a-zA-Z0-
9]*)>(.*)</\\1>','$2').regexReplaceAll('<([a-zA-Z0-9]*)>','')
		.regexReplaceAll('&lt;([a-zA-Z0-
9]*)>(.*)&lt;/\\1>','$2').regexReplaceAll('&lt;([a-zA-Z0-9]*)>','')

	.regexReplaceAll('\\\\(.)','$1').regexReplaceAll('&','&amp;').regexRep
laceAll('"','&quot;');

rule definition {
  from s: RWM!RequirementsDefinition
  to t : MindMap!Maps (
	maps <- RWM!Agent->allInstances()
	)
}

rule agent {
	from s : RWM!Agent
	to t : MindMap!"Map" (
	  node <- t_node
	),
	t_node : MindMap!Node (
	  text <- s.name+' - Objectifs',
	  sub <- s.isResponsible->collect(goal |
thisModule.createNodeFromGoal(goal))
	)
}

rule createNodeFromGoal(s : RWM!Goal) {
	to t : MindMap!Node (
		text <- s.name,
		sub <- Sequence{description}->union(s.subgoals->collect(goal |
thisModule.createNodeFromGoal(goal)))
	),
	description : MindMap!Node (
		text <-
			if (s.documentation <> OclUndefined) then
				s.documentation.normalize()
			else
				''
			endif,
		presentation <- font_italic
	),
	font_italic : MindMap!NodePresentation (
		bold <- false,
		italic <- true,
		size <- 10,
		name <- 'SansSerif'
	)
	do {
```

```
                t;
        }
}
```

Listing 9 : Transformation ATL Req2Risk

```
-- @atlcompiler atl2006
module RWM2Risk; -- Module Template
create OUT : RISK from IN : RWM;

------------------------------
------------------------------

helper context RWM!Goal def : priorityAsDouble : Real =
    if (self.priority = #VeryHigh) then
      4.0
    else if (self.priority = #High) then
      3.0
    else if (self.priority = #Normal) then
      2.0
    else if (self.priority = #Low) then
      1.0
    else if (self.priority = #VeryLow) then
      0.0
    else
      2.0 --By default
       endif endif endif endif endif;

helper def : maxPriority() : Integer = 4;

--Goal coefficient
helper def : unitary_coeff() : Real = 1.0;
helper def : parent_coeff() : Real = 1.0;
helper def : child_coeff() : Real = 0.5;

--Information system coefficient
helper def : si_create_coeff() : Real = 0.5;
helper def : si_read_coeff() : Real = 1.0;
helper def : si_update_coeff() : Real = 1.0;
helper def : si_delete_coeff() : Real = 0.5;

helper context RWM!Goal def : getAllSubGoals() : Sequence(RWM!Goal) =
      Sequence{self}->union(self.subgoals->collect(goal |
goal.getAllSubGoals())->flatten());

helper context RWM!Goal def : entitiesByGoal() : Sequence(RWM!Entity) =
    self.privilegeGroup->flatten()->select(elt | elt <> OclUndefined)-
>collect(pGroup | pGroup.privileges)->flatten()->collect(p | p.element)-
>asSet()->select(e | e.oclIsTypeOf(RWM!Entity));

helper context RWM!Goal def : attributesByGoal() : Sequence(RWM!Attribute)
=
    self.privilegeGroup->flatten()->select(elt | elt <> OclUndefined)-
>collect(pGroup | pGroup.privileges)->flatten()->collect(p | p.element)-
>asSet()->select(e | e.oclIsTypeOf(RWM!Attribute));

helper context RWM!Goal def : risk : Real =
    let entityCoverage : Real = self.entitiesByGoal().size() / RWM!Entity-
>allInstances()->size()
      in let attributeCoverage : Real = self.attributesByGoal().size() /
```

```
RWM!Attribute->allInstances()->size()
      in let goalPriority : Real = self.priorityAsDouble /
thisModule.maxPriority()
    in
      (entityCoverage + attributeCoverage + goalPriority)/3.0;

helper context RWM!BasicElement def : getPrivileges(nature :
RWM!PrivilegeNature) : Sequence(RWM!Privilege) =
    RWM!Privilege->allInstances()->select(p | p.element=self and p.category
= nature);

helper def : getPrivileges(nature : RWM!PrivilegeNature, type : OclType) :
Sequence(RWM!Privilege) =
    RWM!Privilege->allInstances()->select(p | p.category = nature and
p.element.oclIsTypeOf(type));

helper context RWM!Entity def : risk : Real =
    let rge : Real =
            thisModule.getPrivileges(#"create",RWM!Entity)-
>size()*thisModule.si_create_coeff() +
            thisModule.getPrivileges(#read,RWM!Entity)-
>size()*thisModule.si_read_coeff() +
            thisModule.getPrivileges(#update,RWM!Entity)-
>size()*thisModule.si_update_coeff() +
            thisModule.getPrivileges(#delete,RWM!Entity)-
>size()*thisModule.si_delete_coeff()
      in
      let re : Real =
            self.getPrivileges(#"create")-
>size()*thisModule.si_create_coeff() +
            self.getPrivileges(#read)->size()*thisModule.si_read_coeff()
+
            self.getPrivileges(#update)-
>size()*thisModule.si_update_coeff() +
            self.getPrivileges(#delete)-
>size()*thisModule.si_delete_coeff()
      in
      let rga : Real =
            thisModule.getPrivileges(#"create",RWM!Attribute)-
>size()*thisModule.si_create_coeff() +
            thisModule.getPrivileges(#read,RWM!Attribute)-
>size()*thisModule.si_read_coeff() +
            thisModule.getPrivileges(#update,RWM!Attribute)-
>size()*thisModule.si_update_coeff() +
            thisModule.getPrivileges(#delete,RWM!Attribute)-
>size()*thisModule.si_delete_coeff()
      in
      let raset : Sequence(Real) =
          self.attributes->collect( a |
            a.getPrivileges(#"create")-
>size()*thisModule.si_create_coeff() +
            a.getPrivileges(#read)->size()*thisModule.si_read_coeff() +
            a.getPrivileges(#update)->size()*thisModule.si_update_coeff()
+
            a.getPrivileges(#delete)->size()*thisModule.si_delete_coeff()
            )
      in
```

```
    if (raset->size() = 0) then
            re / rge
    else
        ((re / rge) + (raset->sum() / rga))/2
     endif;

helper context RWM!Agent def : risk : Real =
    let allGoals : Sequence(RWM!Goal) =
        self.isResponsible->collect(g | g.getAllSubGoals())->flatten()-
>asSet()
    in
      allGoals.size() / RWM!Goal->allInstances()->size();

------------------------------
------------------------------

rule reqDefs {
    from s : RWM!RequirementsDefinition
    to t : RISK!RiskAnalysis (
        estimation <- RISK!RiskEstimation->allInstances()
    )
}

rule goal {
    from s : RWM!Goal
    to t : RISK!RiskEstimation (
        elementName <- s.name,
        elementType <- 'Goal',
        value <- s.risk
    )
}

rule entity {
    from s : RWM!Entity
    to t : RISK!RiskEstimation (
        elementName <- s.name,
        elementType <- 'Entity',
        value <- s.risk
    )
}

rule agent {
    from s : RWM!Agent
    to t : RISK!RiskEstimation (
        elementName <- s.name,
        elementType <- 'Agent',
        value <- s.risk
    )
}
```

Listing 10 : Transformation ATL Req2Prototype

```
-- @atlcompiler atl2006
module RWM2WebProject; -- Module Template
create OUT : WebProject from IN : RWM;

helper def : webProject : WebProject!WebProject = OclUndefined;
helper def : projectSchema : WebProject!Schema = OclUndefined;

helper context RWM!Goal def : getAllSubGoals() : Sequence(RWM!Goal) =
      Sequence{self}->union(self.subgoals->collect(goal |
goal.getAllSubGoals())->flatten());

helper context String def : normalize() : String =
    self.regexReplaceAll('[^a-zA-Z]','');

helper context String def : format() : String =

self.regexReplaceAll('&','&amp;').regexReplaceAll('<','&lt;').regexReplaceAl
l('>','&gt;')
             .regexReplaceAll('\\n','<br/>');

rule ReqDefs {
    from s : RWM!RequirementsDefinition
    to t : WebProject!Project (
        name <- 'My Web Project',
        pages <- WebProject!Page->allInstances(),
        schema <- schema
    ),
    schema : WebProject!Schema(
        tables <- WebProject!Table->allInstances()
    ),
    loginP : WebProject!LoginPage (
        title <- s.name,
        name <- 'index.html',
             id <- s.id,
             comment<-Sequence{s.documentation.format()},
        links <- RWM!Agent->allInstances()->collect(a |
thisModule.resolveTemp(a,'link_loginPage'))->
                                union(RWM!Process->allInstances()-
>collect(a | thisModule.resolveTemp(a,'link_loginPage')))
    )
    do {
        thisModule.webProject <- t;
        thisModule.projectSchema <- schema;
    }
}

rule agent {
    from s : RWM!Agent
    to
    link_loginPage : WebProject!Link(
        label <- s.name,
        page <- goalPage
    ),
    framePage : WebProject!FramePage(
```

```
        name <- 'frame_'+s.__xmiID__+'.html',
        title <- thisModule.webProject.name,
        col1 <- goalPage
    ),
    goalPage : WebProject!GoalPage (
        name <- s.__xmiID__+'.html',
        title <- 'Goals of '+s.name,
             id <- s.id,
        items <- s.isResponsible->collect(g | thisModule.createGoalItem(g))
    )
}

rule createGoalItem(s : RWM!Goal) {
      to t : WebProject!GoalItem (
             label <- s.name,
             page <- thisModule.resolveTemp(s,'dataPage'),
             sub <- s.subgoals->collect(g | thisModule.createGoalItem(g))
      )
      do {
             t;
      }
}

rule createNextGoalItem(s : RWM!Goal, p: RWM!Process) {
      to t : WebProject!GoalItem (
             label <- s.name,
             page <- thisModule.resolveTemp(s,'dataPage'),
             sub <- s.step->select(st | st.process = p)->collect(st |
st.nextGoals)->flatten()->asSet()->collect(g |
thisModule.createNextGoalItem(g,p))
      )
      do {
             t;
      }
}

helper def : sortEntities(seq : Sequence(RWM!Entity), pset :
Sequence(RWM!Privilege), first : RWM!Entity) : Sequence(RWM!Entity) =
      if (seq->size() <= 1) then
             seq
      else
      let newpset : Sequence(RWM!Privilege) =
           pset->select(p | p.element.oclIsTypeOf(RWM!RelationShip))-
>select(p | p.element.source = first or p.element.target = first)
       in
       let rset : Sequence(RWM!RelationShip) =
           newpset->collect(p | p.element)
       in
       let entityset : Set(RWM!Entity) =
           rset->collect(r | r.source)->union(rset->collect(r | r.target))-
>flatten()->excluding(first)->asSet()
       in
       let next : RWM!Entity =
            if (entityset.size() >= 1) then
                  entityset->asSequence()->first()
            else
                seq->first()
```

```
            endif
        in
            thisModule.sortEntities(seq->excluding(first),pset-
>excluding(newpset),next)->prepend(first)
       endif;

rule goalWithPrivileges {
    from s : RWM!Goal (s.privilegeGroup.size() > 0)
    to dataPage : WebProject!DataPage (
        id <- s.id,
            comment<-
Sequence{s.documentation.format(),s.synopsis.format()},
       title <- s.name,
        name <- s.__xmiID__+'.php',
        components <-
            if (s.privilegeGroup <> OclUndefined) then
                let allPrivileges : Sequence(RWM!Privilege) =
                    --s.getAllSubGoals()->collect(g | g.privilegeGroup)-
>select(pg | pg <> OclUndefined)->collect(pg | pg.privileges)->flatten()
                    s.privilegeGroup->collect(pg | pg.privileges)-
>flatten()
                in
                let entities : Set(RWM!Entity) =
                      allPrivileges->select(p |
p.element.oclIsTypeOf(RWM!Entity))->collect(p | p.element)->asSet()-
>asSequence()
                in

       thisModule.sortEntities(entities,allPrivileges,s.privilegeGroup-
>first().entryPoint)->collect(e |
thisModule.createComponent(e,allPrivileges))
           else
               Sequence{}
            endif
    )
    do {
        dataPage.mainComponent <- dataPage.components->select(cp | cp.name =
s.privilegeGroup->first().entryPoint.name)->first();
    }
}


rule goalWithoutPrivileges {
    from s : RWM!Goal (s.privilegeGroup.size() = 0)
    to dataPage : WebProject!GoalPage (
       id <- s.id,
            comment<-
Sequence{s.documentation.format(),s.synopsis.format()},
        title <- s.name,
        name <- s.__xmiID__+'.html',
        items <- thisModule.createGoalItem(s)
    )
}

rule createComponent(e : RWM!Entity, pset : Sequence(RWM!Privilege)) {
       to c : WebProject!Component (
          name <- e.name,
```

```
            canCreate <- pset->select(p | p.element = e and p.category =
#"create")->size() > 0,
            canRead <- pset->select(p | p.element = e and p.category = #read)-
>size() > 0,
            canUpdate <- pset->select(p | p.element = e and p.category =
#update)->size() > 0,
            canDelete <- pset->select(p | p.element = e and p.category =
#delete)->size() > 0,
            table <- thisModule.resolveTemp(e,'table'),
            properties <- e.attributes->select(a | pset->select(p | p.element
= a)->size() > 0)->collect(a | thisModule.createComponentAttribute(a,pset))
                ->union(RWM!RelationShip->allInstances()->select(r | r.source =
e or r.target = e)->select(r | pset->select(p | p.element = r)->size() > 0)
                ->collect(r | thisModule.createComponentRelationShip(r,pset)))
    )
    do {
        c;
    }
}

rule createComponentAttribute(a : RWM!Attribute, pset :
Sequence(RWM!Privilege)) {
       to c : WebProject!ComponentAttribute (
            name <- a.name,
            field <- thisModule.resolveTemp(a,'field'),
            canRead <- pset->select(p | p.element = a and p.category = #read)-
>size() > 0,
            canUpdate <- pset->select(p | p.element = a and p.category =
#update)->size() > 0
    )
    do {
        c;
    }
}

rule createComponentRelationShip(r : RWM!RelationShip, pset :
Sequence(RWM!Privilege)) {
       to c : WebProject!ComponentRelationShip (
            name <- r.name,
            idLeft <-thisModule.resolveTemp(r.source,'id'),
         manyLeft <- r.sourceMax = -1 or r.sourceMax > 1,
         mandatoryLeft <- r.sourceMin = 1 and r.sourceMax = 1,
            idRight <- thisModule.resolveTemp(r.target,'id'),
         manyRight <- r.targetMax = -1 or r.targetMax > 1,
         mandatoryRight <- r.targetMin = 1 and r.targetMax = 1,
         canRead <- pset->select(p | p.element = r and p.category = #read)-
>size() > 0,
         canUpdate <- pset->select(p | p.element = r and p.category =
#update)->size() > 0
    )
    do {
        c;
    }
}

-----------------------------------------
-- RELATIONAL SCHEMA
```

```
---------------------------------------

rule entity {
    from e : RWM!Entity
    to table : WebProject!Table (
        name <- e.name.normalize(),
        fields <- e.attributes->prepend(id),
        primaryKey <- Sequence{id}
    ),
    id : WebProject!Field (
        name <- 'id_'+e.__xmiID__,
        dataType <- #integer,
        autoincrement <- true
    )
}

rule attribute {
    from a : RWM!Attribute
    to field : WebProject!Field (
        name <- a.name.normalize(),
        dataType <-
              if (a.type = #TextualValue) then
                  #string
              else if (a.type = #NumericalValue) then
                  #real
              else if (a.type = #TemporalValue) then
                  #dateTime
              else
                  #string
              endif endif endif
    )
}

rule relationship {
    from r : RWM!RelationShip (r.source <> OclUndefined and r.target <>
OclUndefined)
    to table : WebProject!Table (
        name <- r.source.name.normalize()+'2'+r.target.name.normalize(),
        fields <- Sequence{idLeft, idRight},
        primaryKey <- Sequence{idLeft,idRight}
    ),
    idLeft : WebProject!Field (
        name <- 'id_'+r.source.__xmiID__,
        dataType <- #integer
    ),
    idRight : WebProject!Field (
        name <- 'id_'+r.target.__xmiID__,
        dataType <- #integer
    )
}

rule process {
    from s : RWM!Process
      using {
              steps : Set(RWM!GoalStep) =
                          RWM!GoalStep->allInstances()->select(step |
step.process = s);
```

```
            goals : Set(RWM!Goal) =
                        steps->collect(s | s.refImmediateComposite())-
>union(steps->collect(s | s.nextGoals)->flatten())->asSet();
            startGoal : Set(RWM!Goal) =
                        goals->select(g | steps->collect(s | s.nextGoals)-
>flatten()->excludes(g));
       }
    to
    link_loginPage : WebProject!Link(
        label <- s.name,
        page <- goalPage
    ),
    framePage : WebProject!FramePage(
        name <- 'frame_'+s.__xmiID__+'.html',
        title <- thisModule.webProject.name,
        col1 <- goalPage
    ),
    goalPage : WebProject!GoalPage (
        name <- s.__xmiID__+'.html',
             id <- s.id,
        title <- 'Process : '+s.name,
        items <- startGoal->collect(g | thisModule.createNextGoalItem(g,s))
    )
}
```

Listing 11 : Transformation ATL Req2Documentation

```
-- @atlcompiler atl2006
module RWM2DocBook; -- Module Template
create OUT : Documentation from IN : RWM;

helper context String def : normalize() : String =
	self.regexReplaceAll('&','&amp;').regexReplaceAll('<','&lt;').regexRe
placeAll('>','&gt;')
		.regexReplaceAll('<([a-zA-Z0-
9]*)>(.*)</\\1>','$2').regexReplaceAll('&lt;([a-zA-Z0-
9]*)>(.*)&lt;/\\1>','$2')

	.regexReplaceAll('\\\\(.)','$1').regexReplaceAll('&#xA;','&#xD;&#xA;')
;


helper context RWM!Goal def : getAllSubGoals() : Sequence(RWM!Goal) =
	Sequence{self}->union(self.subgoals->collect(goal |
goal.getAllSubGoals())->flatten());

helper def : sort(goals : Sequence(RWM!Goal), previous : Set(RWM!Goal), p :
RWM!Process) : Sequence(RWM!Goal) =
	if (goals->size() <= 1) then
		goals
	else
		let first : Sequence(RWM!Goal) =
			previous->collect(g | g.step)->flatten()->select(st |
st.process = p)->collect(st | st.nextGoals)->flatten()->asSet()-
>intersection(goals)
		in
			first->union(thisModule.sort(goals->reject(g |
previous->includes(g) or first->includes(g)),first,p))
	endif;

rule createTextualValue(text : String) {
	to t : Documentation!TextualValue (
		value <- text
	)
	do {
		t;
	}
}

rule createParagraph(text : String) {
	to t : Documentation!Paragraph(
		values <- Sequence{thisModule.createTextualValue(text)}
	)
	do {
		t;
	}
}

rule createSection(_title : String, text : String) {
	to t : Documentation!Section(
```

```
                para <- Sequence{thisModule.createParagraph(text)},
                title <- _title
        )
        do {
                t;
        }
}

rule createPrivilegeSection(s : RWM!Goal) {
        to t_section_privileges:Documentation!Section (
                title <- 'Privilèges',
                para <- Sequence{t_para_privileges}
        ),
        t_para_privileges:Documentation!Paragraph (
                values <- Sequence{s.privilegeGroup}
        )
        do {
                t_section_privileges;
        }
}


rule createLinkItem(element : RWM!BasicElement) {
        to t : Documentation!ItemizedListValueItem(
                values <- Sequence{t_xref}
        ),
        t_xref : Documentation!XRefValue (
                linkend <- element
        )
        do {
                t;
        }
}

rule createLinkItemWithDocumentation(element : RWM!BasicElement) {
        to t : Documentation!ItemizedListValueItem(
                values <- Sequence{t_value,t_xref,t_value2,t_description}
        ),
        t_value : Documentation!TextualValue (
                value <- element.name.normalize() +' ( voir'
        ),
        t_xref : Documentation!XRefValue (
                linkend <- element
        ),
        t_value2 : Documentation!TextualValue (
                value <- ')'
        ),
        t_description : Documentation!EmphasisValue (
                value <-
                        if (element.documentation <> OclUndefined) then
                                ' : ' + element.documentation.normalize()
                        else
                                ''
                        endif
        )
        do {
                t;
```

```
	}
}

rule createLinkList(elements : Sequence(RWM!BasicElement)) {
	to t : Documentation!ItemizedListValue(
		items <- elements->collect(e | thisModule.createLinkItem(e))
	)
	do {
		t;
	}
}

rule createLink(element : RWM!BasicElement) {
	to t : Documentation!Paragraph(
		values <- Sequence{t_value,t_xref,t_value2,t_description}
	),
	t_value : Documentation!TextualValue (
		value <- element.name +' ( voir'
	),
	t_xref : Documentation!XRefValue (
		linkend <- element
	),
	t_value2 : Documentation!TextualValue (
		value <- ')'
	),
	t_description : Documentation!EmphasisValue (
		value <-
			if (element.documentation <> OclUndefined) then
				'(' + element.documentation.normalize() + ')'
			else
				'()'
			endif
	)
	do {
		t;
	}
}

rule createLinkWithoutParagraph(element : RWM!BasicElement) {
	to t_xref : Documentation!XRefValue (
		linkend <- element
	)
	do {
		Sequence{t_xref};
	}
}

rule createProcessStep(goal : RWM!Goal, p : RWM!Process, start : Boolean) {
	using {
		nextGoals : Sequence(RWM!Goal) =
			goal.step->select(st | st.process = p)->collect(st |
st.nextGoals)->flatten()->asSet();
	}
	to t : Documentation!ItemizedListValueItem(
		values <- Sequence{t_value,t_xref,t_value2,t_value3}
	),
	t_value : Documentation!TextualValue (
```

```
		value <-
			if (start) then
				'[DEBUT] '
			else
				''
			endif
	),
	t_xref : Documentation!XRefValue (
		linkend <- goal
	),
	t_value2 : Documentation!TextualValue (
		value <-
			if (nextGoals->size() > 0) then
				'\n Prochaine(s) étape(s) : '
			else
				''
			endif
	),
	t_value3 : Documentation!ItemizedListValue(
		items <- nextGoals->collect(g |
thisModule.createProcessNextStep(g))
	)
	do {
		t;
	}
}

rule createProcessNextStep(goal : RWM!Goal) {
	to t : Documentation!ItemizedListValueItem(
		values <- Sequence{t_value}
	),
	t_value : Documentation!TextualValue (
		value <- goal.name.normalize()
	)
	do {
		t;
	}
}

rule ReqDefs {
	from s:RWM!RequirementsDefinition
	to t:Documentation!Book (
		title <- s.name+'

		Cahier des charges',
		content <-
			let preface : Documentation!Section =
				thisModule.createSection('Préface','Le but de ce
document est de décrire les besoins métier de l\'application cible.
Il s\'appuie sur une méthodologie de
description des besoins basée sur les objectifs.
Cette méthodologie consiste à:
1/ identifier chaque catégorie
d\'acteur pouvant intervenir sur l\'application,
2/ définir par catégorie d\'acteur
les objectifs métier qu\'il souhaite adresser via l\'application et dont on
dit qu\'il est responsable.
```

```
                                        3/ structurer ces objectifs soit en
termes de processus pour tenir compte de la dépendance des objectifs, soit
en termes de hiérarchies pour tenir compte de la spécialisation progressive
d\'objectifs (hiérarchie de sous-objectifs).
                                        4/ caractériser ces objectifs à
travers les objets que l\'acteur va être amené à manipuler pour adresser ses
objectifs via l\'application. Par exemple, un objet peut être un document
qui sera manipulé dans le cadre d\'un objectif de "publication de
documents". Ces objets sont décrits à travers leurs attributs et les
relations qu\'ils ont entre eux.

                            Cette méthodologie est mise en oeuvre dans
l\'environnement MDA SIDE de BlueXML à travers des modèles de besoins créé
sous le modeleur graphique de besoins: ces modèles sont ensuite générés pour
produire un certain nombre de livrables dont ce cahier des charges orienté
objectifs. Un autre livrable est une application web d\'annotations
permettant à des fonctionnels d\'annoter les modèles pour les améliorer et
mieux circonscrire les besoins.')
                    in let structure : Documentation!Section =
                            thisModule.createSection('Structure de la
documentation','Ce document est structuré en 4 chapitres principaux:
                                    - "Agents": ce chapitre liste les
catégories d\'acteurs visant à utiliser l\'application cible et pour
lesquels il est nécessaire de décrire les besoins adressés,
                                    - "Objectifs": ce chapitre décrit les
caractéristiques des objectifs adressés,
                                    - "Processus": ce chapitre décrit des
processus métier particuliers de l\'application faisant apparaître
l\'enchainement d\'objectifs dépendants,
                                    - "Entités": ce chapitre décrit les objets
manipulés par les objectifs, les privilèges en terme de manipulation
(création, visualisation, mise à jour et suppression) des ces objets par les
acteurs et les relations entre ces objets.')
                    in let project : Documentation!Section =
                            thisModule.createSection('Définition du
projet',s.documentation.normalize())
                    in

        Sequence{preface,structure,project,t_chapter_agents,t_chapter_goals,t_
chapter_process,t_chapter_entities}
        ),
        t_chapter_agents:Documentation!Section (
                title <- 'Agents',
                section <- RWM!Agent->allInstances()
        ),
        t_chapter_goals:Documentation!Section (
                title <- 'Objectifs',
                section <- RWM!Goal->allInstances()
        ),
        t_chapter_entities:Documentation!Section (
                title <- 'Entités',
                section <- RWM!Entity->allInstances()
        ),
        t_chapter_process:Documentation!Section (
                title <- 'Processus',
                section <- RWM!Process->allInstances()
        )
```

```
}

rule Member {
	from s:RWM!Agent
	to t:Documentation!Section (
		title <- s.name,
		section <- Sequence{t_section_description,
t_section_responsibilities,t_section_privileges}
	),
	t_section_description:Documentation!Section (
		title <- 'Documentation',
		para <- Sequence{t_para_description,
t_para_description_isHuman}
	),
	t_para_description:Documentation!Paragraph (
		values <- Sequence{t_para_description_value}
	),
	t_para_description_value:Documentation!TextualValue (
		value <-
			if (s.documentation <> OclUndefined) then
				s.documentation.normalize()
			else
				''
			endif
	),
	t_para_description_isHuman:Documentation!Paragraph (
		values <-
Sequence{t_para_description_isHuman_value1,t_para_description_isHuman_value2
}
	),
	t_para_description_isHuman_value1:Documentation!EmphasisValue (
		value <- 'Est un agent humain : ',
		role <- 'bold'
	),
	t_para_description_isHuman_value2:Documentation!EmphasisValue (
		value <- if (s.isHuman) then
				'Oui'
			else
				'Non'
			endif
	),
	t_section_responsibilities:Documentation!Section (
		title <- 'Responsabilité(s)',
		para <- Sequence{t_para_responsibilities}
	),
	t_para_responsibilities:Documentation!Paragraph (
		values <-
Sequence{t_para_responsibilities_list_value,t_para_responsibilities_list}
	),
	t_para_responsibilities_list_value:Documentation!TextualValue (
		value <- 'A la responsabilité de : '
	),
	t_para_responsibilities_list:Documentation!ItemizedListValue (
		items <- s.isResponsible->collect(g |
thisModule.createLinkItem(g))
	),
```

```
	t_section_privileges:Documentation!Section (
		title <- 'Privilège(s)',
		para <- Sequence{t_para_privileges}
	),
	t_para_privileges:Documentation!Paragraph (
		values <- Sequence{t_para_privileges_value}
	),
	t_para_privileges_value:Documentation!InformalTableValue (
		cols <- 5,
		headRows <- Sequence {t_row},
		bodyRows <- let allPrivileges : Sequence(RWM!BasicElement) =
				s.isResponsible->collect(goal |
goal.getAllSubGoals())->flatten()->collect(goal | goal.privilegeGroup)-
>excluding(OclUndefined)->flatten()->collect(pGroup | pGroup.privileges)-
>flatten()
				in
				let allPrivilegesElement :
Sequence(RWM!BasicElement) =
				allPrivileges->collect(p | p.element)->flatten()
					in
						let	entities :
Sequence(RWM!Entity) =
							allPrivilegesElement->select(e
| e.oclIsTypeOf(RWM!Entity))->
						union( allPrivilegesElement->select(e
| e.oclIsTypeOf(RWM!Attribute))->collect(e | e.refImmediateComposite()))->
						asSet()->asSequence()
					in
						entities->iterate(entity; all :
Sequence(RWM!BasicElement) = Sequence{} | all->append(entity)-
>union(entity.attributes)->union(RWM!RelationShip->allInstances()->select(r
| r.source = entity or r.target = entity)))
						->collect(e |
thisModule.Privilege(e,allPrivileges,OclUndefined))
	),
	t_row:Documentation!InformalTableValueRow (
		entry <- Sequence{t_entry_element, t_entry_create,
t_entry_read, t_entry_update, t_entry_delete}
	),
	t_entry_element:Documentation!TextualValue(
		value <- 'Entité &amp;amp; attribut'
	),
	t_entry_create:Documentation!TextualValue(
		value <- 'Créer'
	),
	t_entry_read:Documentation!TextualValue(
		value <- 'Lire'
	),
	t_entry_update:Documentation!TextualValue(
		value <- 'Mettre a jour'
	),
	t_entry_delete:Documentation!TextualValue(
		value <- 'Supprimer'
	)
}

rule Goal {
```

```
	from s:RWM!Goal
	to t:Documentation!Section (
		title <- s.name.normalize(),
		section <-
			if (not(s.synopsis.oclIsUndefined())) then
				if (s.privilegeGroup->collect(pg |
pg.privileges)->flatten()->size() > 0) then

	Sequence{t_section_description,thisModule.createSection('Synopsis',s
.synopsis.normalize()),t_section_responsibilities,thisModule.createPrivileg
eSection(s)}
				else

	Sequence{t_section_description,thisModule.createSection('Synopsis',s
.synopsis.normalize()),t_section_responsibilities}
				endif
			else
				if (s.privilegeGroup->collect(pg |
pg.privileges)->flatten()->size() > 0) then

	Sequence{t_section_description,t_section_responsibilities,thisModule
.createPrivilegeSection(s)}
				else

	Sequence{t_section_description,t_section_responsibilities}
				endif
			endif
	),
	t_section_description:Documentation!Section (
		title <- 'Documentation',
		para <-
Sequence{t_para_description,t_para_description_level,t_para_description_pare
nt,t_para_description_child}
	),
	t_para_description:Documentation!Paragraph (
		values <- Sequence{t_para_description_value}
	),
	t_para_description_value:Documentation!TextualValue (
		value <-
			if (s.documentation <> OclUndefined) then
				s.documentation.normalize()
			else
				''
			endif
	),
	t_para_description_level:Documentation!Paragraph (
		values <-
Sequence{t_para_description_level_value1,t_para_description_level_value2}
	),
	t_para_description_level_value1:Documentation!EmphasisValue (
		value <- 'Niveau d\'importance : ',
		role <- 'bold'
	),
	t_para_description_level_value2:Documentation!EmphasisValue (
		value <-
			if (s.priority = #VeryHigh) then
				'Très important'
```

```
                    else if (s.priority = #High) then
                          'Important'
                    else if (s.priority = #Normal) then
                          'Normal'
                    else if (s.priority = #Low) then
                          'Faible'
                    else if (s.priority = #VeryLow) then
                          'Très faible'
                    else
                          ''
                    endif endif endif endif endif
      ),
      t_para_description_parent:Documentation!Paragraph (
            values <- if (RWM!Goal->allInstances()->select(g | g.subgoals-
>includes(s))->size() > 0) then
                        thisModule.createLinkWithoutParagraph(RWM!Goal-
>allInstances()->select(g | g.subgoals->includes(s))->first())-
>insertAt(1,t_para_description_parent_value)
                         else
                              Sequence{t_para_description_parent_value,
thisModule.createTextualValue('N/A')}
                    endif
      ),
      t_para_description_parent_value:Documentation!EmphasisValue (
            value <- 'Objectif parent : ',
            role <- 'bold'
      ),
      t_para_description_child:Documentation!Paragraph (
            values <- if (s.subgoals->size() > 0) then

Sequence{t_para_description_child_list_value,thisModule.createLinkList(s.su
bgoals)}
                    else
                        Sequence{t_para_description_child_list_value,
thisModule.createTextualValue('N/A')}
                    endif
      ),
      t_para_description_child_list_value:Documentation!EmphasisValue (
            value <- 'Liste des sous-objectifs : ',
            role <- 'bold'
      ),
      t_section_responsibilities:Documentation!Section (
            title <- 'Responsabilité(s)',
            para <- Sequence{t_para_responsibilities}
      ),
      t_para_responsibilities:Documentation!Paragraph (
            values <- if (s.subgoals->size() > 0) then

Sequence{t_para_responsibilities_list_value,thisModule.createLinkList(s.res
ponsible)}
                    else
                        Sequence{t_para_responsibilities_list_value,
thisModule.createTextualValue('N/A')}
                    endif
      ),
      t_para_responsibilities_list_value:Documentation!TextualValue (
            value <- 'Sous la responsabilité de : '
```

```
	)
}

rule PrivilegeGroup {
	from s:RWM!PrivilegeGroup(s.privileges.size() > 0)
	to t:Documentation!InformalTableValue (
		cols <- 5,
		headRows <- Sequence {t_row},
		bodyRows <- let entities : Sequence(RWM!Entity) =
				s.privileges->collect(p | p.element)->select(e |
e.oclIsTypeOf(RWM!Entity))->
				union( s.privileges->collect(p | p.element)-
>select(e | e.oclIsTypeOf(RWM!Attribute))->collect(e |
e.refImmediateComposite()) )->
				asSet()->asSequence()
				in
				  entities->iterate(entity; all :
Sequence(RWM!BasicElement) = Sequence{} | all->append(entity)-
>union(entity.attributes)->union(RWM!RelationShip->allInstances()->select(r
| r.source = entity or r.target = entity)))
				  ->collect(e |
thisModule.Privilege(e,s.privileges,s.entryPoint))
	),
	t_row:Documentation!InformalTableValueRow (
		entry <- Sequence{t_entry_element, t_entry_create,
t_entry_read, t_entry_update, t_entry_delete}
	),
	t_entry_element:Documentation!TextualValue(
		value <- 'Entité &amp;amp; attribut'
	),
	t_entry_create:Documentation!TextualValue(
		value <- 'Créer'
	),
	t_entry_read:Documentation!TextualValue(
		value <- 'Lire'
	),
	t_entry_update:Documentation!TextualValue(
		value <- 'Mettre a jour'
	),
	t_entry_delete:Documentation!TextualValue(
		value <- 'Supprimer'
	)
}

rule Privilege(element : RWM!BasicElement, group : Sequence(RWM!Privilege),
entryPoint : RWM!BasicElement)  {
	to t:Documentation!InformalTableValueRow (
	  entry <- Sequence{t_entry_element, t_entry_create, t_entry_read,
t_entry_update, t_entry_delete}
	),
	t_entry_element:Documentation!TextualValue(
		value <- if (element.oclIsTypeOf(RWM!Entity)) then
				if (element = entryPoint) then
					'&amp;gt; '+element.name
				else
				element.name
				endif
```

```
                        else
                          if (element.oclIsTypeOf(RWM!Attribute)) then
                            element.refImmediateComposite().name+' - [A]
'+element.name
                          else
                            '[R] '+element.name
                          endif
                        endif
        ),
        t_entry_create:Documentation!TextualValue(
                value <- if (group->select(p | p.element = element and
p.category = #"create")->size() > 0) then
                        'X'
                      else
                        ''
                        endif
        ),
        t_entry_read:Documentation!TextualValue(
                value <- if (group->select(p | p.element = element and
p.category = #read)->size() > 0) then
                        'X'
                      else
                        ''
                        endif
        ),
        t_entry_update:Documentation!TextualValue(
                value <- if (group->select(p | p.element = element and
p.category = #update)->size() > 0) then
                        'X'
                      else
                        ''
                        endif
        ),
        t_entry_delete:Documentation!TextualValue(
                value <- if (group->select(p | p.element = element and
p.category = #delete)->size() > 0) then
                        'X'
                      else
                        ''
                        endif
        )
        do {
                t;
        }
}

rule Entity {
        from s:RWM!Entity
        to t:Documentation!Section (
                title <- s.name,
                section <-
Sequence{t_section_description,t_section_relationship}
        ),
        t_section_description:Documentation!Section (
                title <- 'Documentation',
                para <- Sequence{t_para_description}
        ),
```

```
	t_para_description:Documentation!Paragraph (
		values <- Sequence{t_para_description_value}
	),
	t_para_description_value:Documentation!TextualValue (
		value <-
			if (s.documentation <> OclUndefined) then
				s.documentation.normalize()
			else
				''
			endif
	),
	t_section_relationship:Documentation!Section (
		title <- 'Relation(s)',
		para <- Sequence{t_para_relationship}
	),
	t_para_relationship:Documentation!Paragraph (
		values <-if (RWM!RelationShip->allInstances()->select(r |
r.source = s or r.target = s)->size() > 0) then

Sequence{t_para_relationship_list_value,thisModule.RelationShip(s)}
				else
				  Sequence{t_para_relationship_list_value}
				endif
	),
	t_para_relationship_list_value:Documentation!TextualValue (
		value <- 'Liste des relations : '
	)
}

rule RelationShip(s : RWM!Entity) {
	to t_para_relationship_list:Documentation!ItemizedListValue (
		items <- RWM!RelationShip->allInstances()->select(r | r.source
= s or r.target = s)->collect(r | Sequence{r.source, r.target})->flatten()-
>asSet()->select(elt | elt <> s)->collect(elt |
thisModule.createLinkItemWithDocumentation(elt))
	)
	do {
		t_para_relationship_list;
	}
}

rule EntityWithAttribute extends Entity {
	from s:RWM!Entity (s.attributes.size() > 0)
	to t:Documentation!Section (
		section <-
Sequence{t_section_description,t_section_attributes,t_section_relationship}
	),
	t_section_attributes:Documentation!Section (
		title <- 'Attribut(s)',
		para <- Sequence{t_para_attributs}
	),
	t_para_attributs:Documentation!Paragraph (
		values <- Sequence{t_para_attributs_table}
	),
	t_para_attributs_table:Documentation!InformalTableValue (
		cols <- 3,
		headRows <- Sequence {t_row},
```

```
                bodyRows <- s.attributes
        ),
        t_row:Documentation!InformalTableValueRow (
                entry <- Sequence{t_entry_name,
t_entry_type,t_entry_description}
        ),
        t_entry_name:Documentation!TextualValue(
                value <- 'Nom'
        ),
        t_entry_type:Documentation!TextualValue(
                value <- 'Type'
        ),
        t_entry_description:Documentation!TextualValue(
                value <- 'Description'
        )
}

rule Attribute {
        from s:RWM!Attribute
        to t:Documentation!InformalTableValueRow (
          entry <- Sequence{t_entry_name, t_entry_type,t_entry_description}
        ),
        t_entry_name:Documentation!TextualValue(
                value <- s.name
        ),
        t_entry_type:Documentation!TextualValue(
                value <- s."type".toString()
        ),
        t_entry_description:Documentation!TextualValue(
                value <-
                        if (s.documentation <> OclUndefined) then
                                s.documentation.normalize()
                        else
                                ''
                        endif
        )
}

rule Process {
    from s : RWM!Process
        using {
                steps : Set(RWM!GoalStep) =
                                RWM!GoalStep->allInstances()->select(step |
step.process = s);
                goals : Set(RWM!Goal) =
                                steps->collect(s | s.refImmediateComposite())-
>union(steps->collect(s | s.nextGoals)->flatten())->asSet();
                startGoal : Set(RWM!Goal) =
                                goals->select(g | steps->collect(s | s.nextGoals)-
>flatten()->excludes(g));
                endGoal : Set(RWM!Goal) =
                                goals->select(g | steps->excludesAll(g.step));
        }
        to t:Documentation!Section (
                title <- s.name.normalize(),
                section <- Sequence{t_section_description,t_section_process}
        ),
```

```
	t_section_description:Documentation!Section (
		title <- 'Documentation',
		para <- Sequence{t_para_description}
	),
	t_para_description:Documentation!Paragraph (
		values <- Sequence{t_para_description_value}
	),
	t_para_description_value:Documentation!TextualValue (
		value <-
			if (s.documentation <> OclUndefined) then
				s.documentation.normalize()
			else
				''
			endif
	),
	t_section_process:Documentation!Section (
		title <- 'Description du processus',
		para <- Sequence{t_para_description_process}
	),
	t_para_description_process:Documentation!Paragraph (
		values <- Sequence{t_para_description_process_list}
	),
	t_para_description_process_list:Documentation!ItemizedListValue (
		items <- startGoal->collect(g |
thisModule.createProcessStep(g,s,true))
			->union(thisModule.sort(goals->select(g | startGoal-
>excludes(g) and endGoal->excludes(g)),startGoal,s)->collect(g |
thisModule.createProcessStep(g,s,false)))
			->union(endGoal->collect(g |
thisModule.createProcessStep(g,s,false)))
	)
}
```